Dubbo 如何成为连接各种异构微服务体系的服务开发框架

刘军（https://github.com/chickenlj）

我们认识中的 Dubbo

0

Service Export

Provider export the services and listen on specific TCP port.

1

Service Registry

Provider register service info to registry: ip, port and other metadata

2

Service Subscribe

Consumer subscribe to services it cares, and try pull service list first.

3

Service Discovery

When service addresses change, registry notify Consumer of new address list.

4

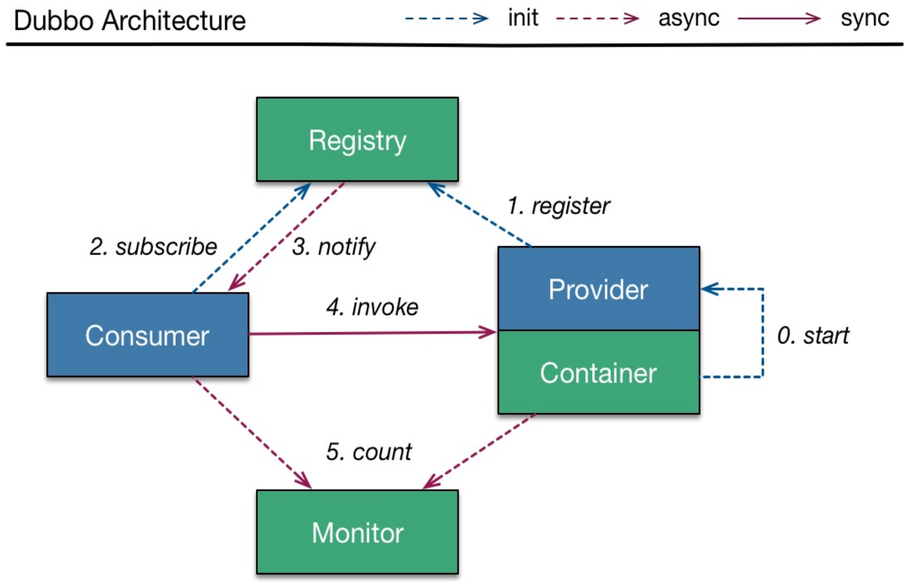
Service Invoke (RPC)

Pick a service provider according to loadbalance and routing policy, invoke directly.

5

Monitor

Statistics between consumer and provider are collected and displayed by Monitor

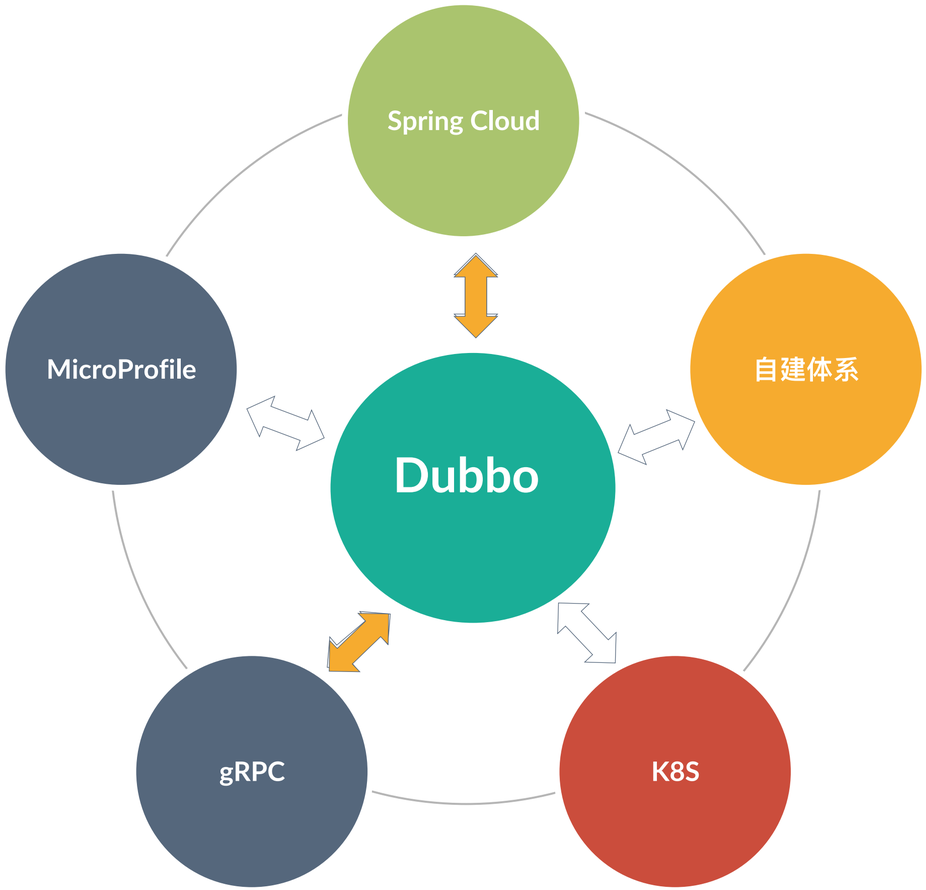


今天要讲的 Dubbo

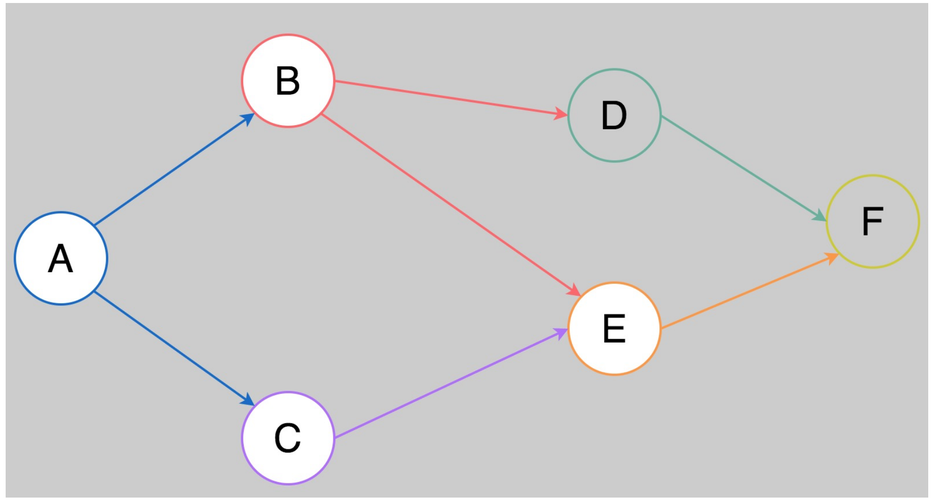
Dubbo 如何成为连接各种异构微服务体系的最佳服务开发框架

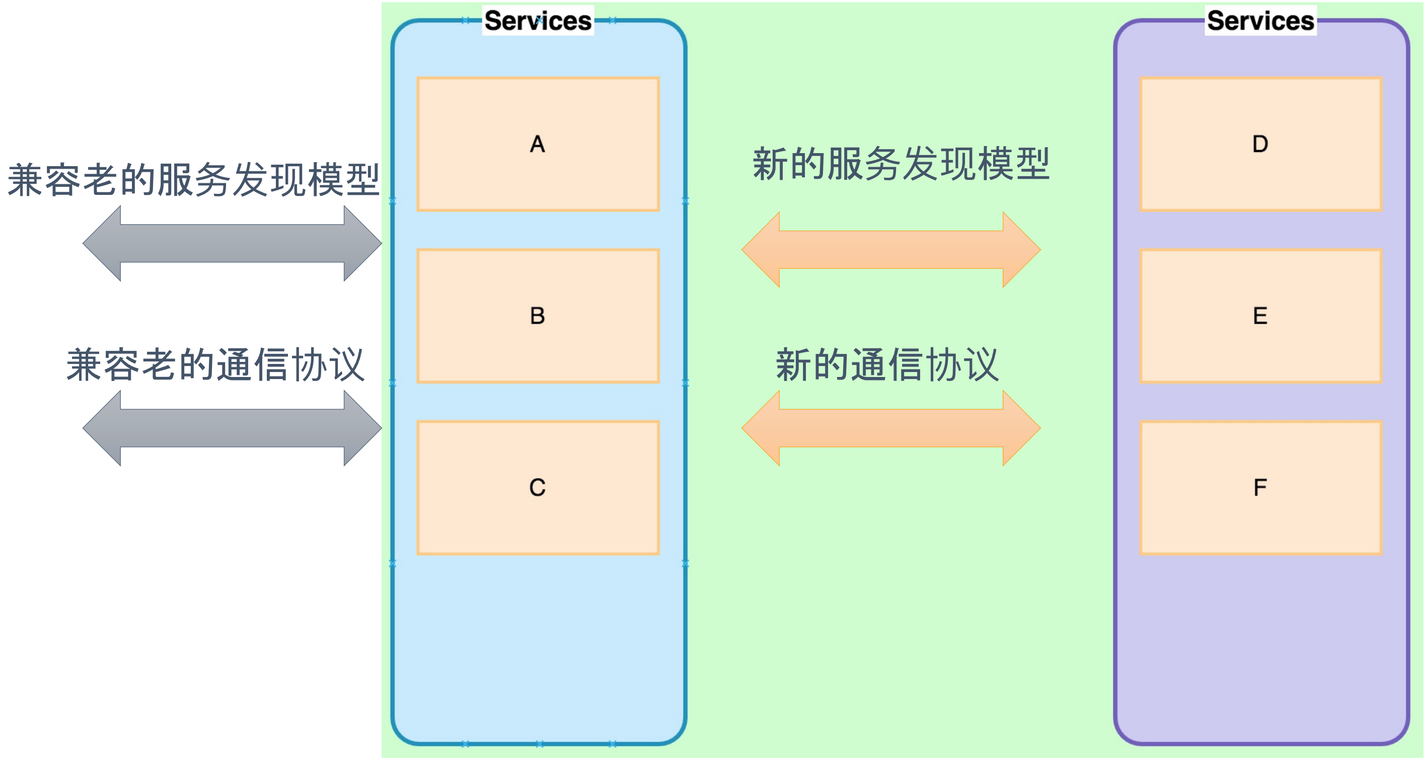
1. 提供透明的面向接口代理的编程模型
2. 协议平滑升级/迁移
3. 异构体系服务互发现

异构微服务体系



场景示例





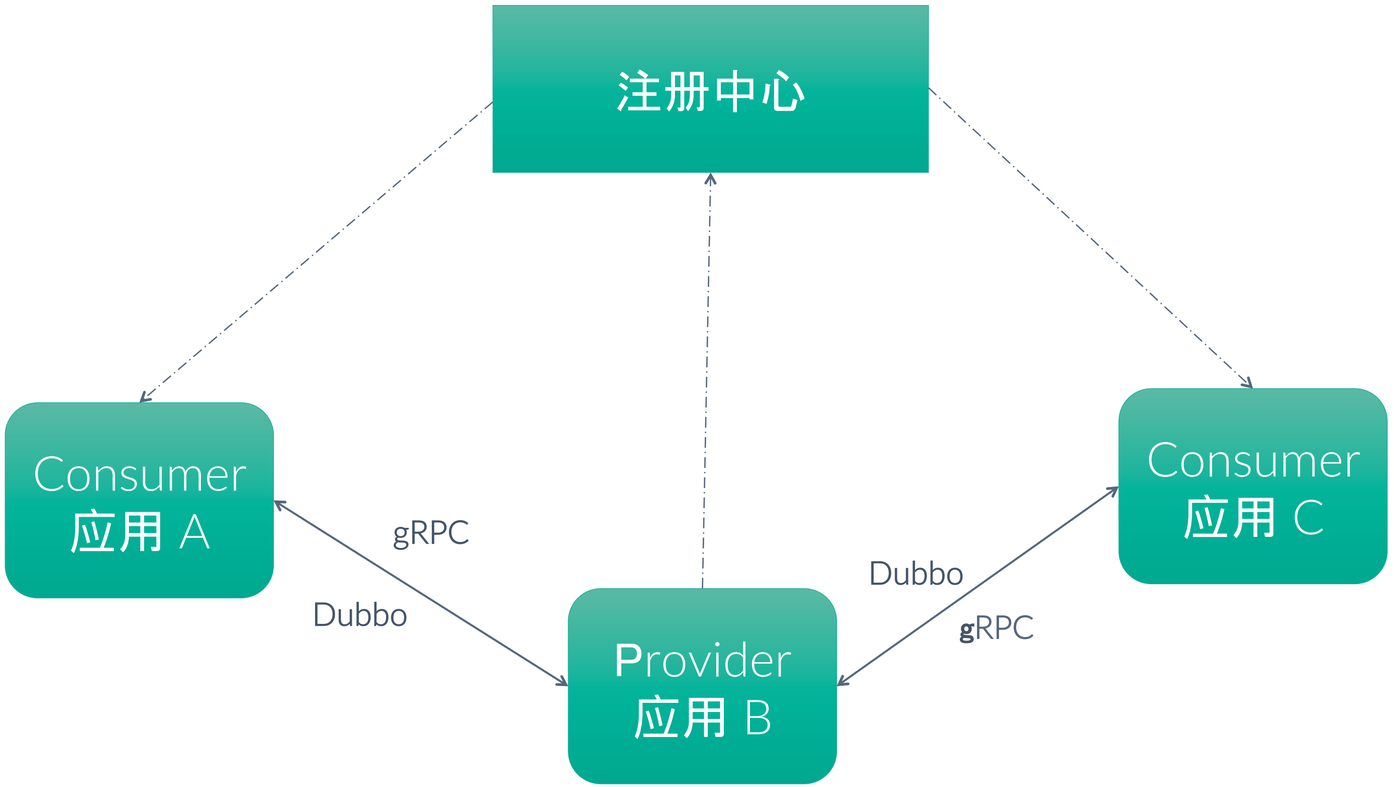
目录

1. 多协议 多注册机制

多协议交互

Service 1
Service 2

Service 0



Service 3
Service 4

Service 0

发布不同协议的服务：

1. Service 0 **同时发布** Dubbo **和** gRPC **协议**
2. Service 1 2 **发布** Dubbo **协议**
3. Service 3 4 **发布** gRPC **协议**

多协议示例

1. 应用 B，提供服务

```
<dubbo:service interface="org.apache.dubbo.samples.basic.api.DemoService1" protocol="dubbo" ref="demoService"/>
<dubbo:service interface="org.apache.dubbo.samples.basic.api.DemoService2" protocol="dubbo" ref="demoService"/>

<dubbo:service interface="org.apache.dubbo.samples.basic.api.DemoService3" protocol="grpc" ref="demoService"/>
<dubbo:service interface="org.apache.dubbo.samples.basic.api.DemoService4" protocol="grpc" ref="demoService"/>

<dubbo:service interface="org.apache.dubbo.samples.basic.api.DemoService0" protocol=”dubbo, grpc"
ref="demoService"/>
```

多协议示例

2. 应用 A，消费服务

```
<dubbo:reference protocol="dubbo" interface="org.apache.dubbo.samples.basic.api.DemoService1"/>
<dubbo:reference protocol="dubbo" interface="org.apache.dubbo.samples.basic.api.DemoService2"/>

<dubbo:reference protocol=”grpc" interface="org.apache.dubbo.samples.basic.api.DemoService0"/>
```

3. 应用 C，消费服务

```
<dubbo:reference protocol=”grpc" interface="org.apache.dubbo.samples.basic.api.DemoService3"/>
<dubbo:reference protocol=”grpc" interface="org.apache.dubbo.samples.basic.api.DemoService4"/>

<dubbo:reference protocol=”dubbo" interface="org.apache.dubbo.samples.basic.api.DemoService0"/>
```

多协议解决的问题

接入 Dubbo 服务治理体系

通过协议扩展将 RPC 协议纳入 Dubbo 服务开发体系，从而复用 Dubbo 的编程模型和服务发现、流量管控等能力。比如 gRPC，其服务治理体系相对比较弱、编程 API 不够友好，很难直接用于微服务开发。

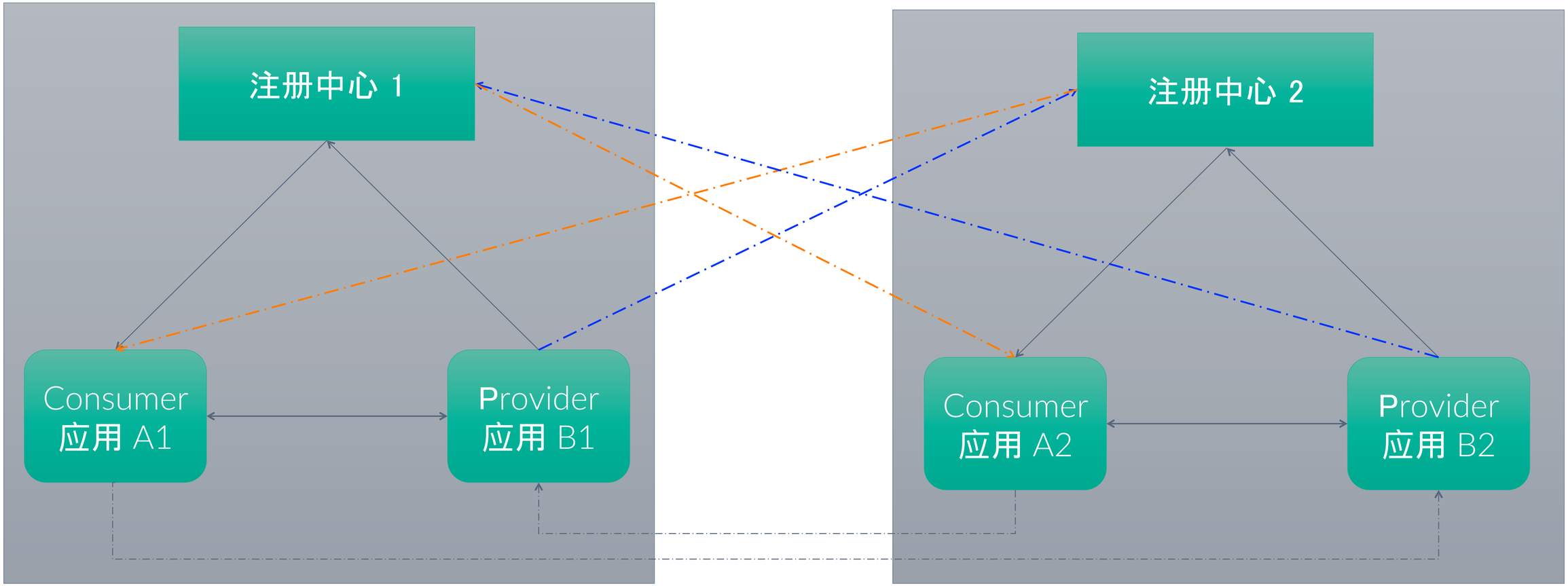
满足不同场景的调用需求

各个服务可能是为了满足不同业务需求而开发，同时外围消费端应用的技术栈也可能多种多样，通过启用不同的通信协议，可以最优化不同场景的通信需求。

协议迁移

通过支持多种协议，借助注册中心的协调，可以快速满足公司内协议迁移的需求。如从自有协议升级到 Dubbo 协议，Dubbo 协议自身升级，从 Dubbo 协议迁移到 gRPC，从 REST 迁移到 Dubbo 协议等。

多注册中心



- 注册中心数据同步 ✖
- 使用同一个注册中心 ✖

1. 提供端，多注册

```
<dubbo:registry id="beijingRegistry" address="zookeeper://${zookeeper.address1}" default="false”/>
<dubbo:registry id="shanghaiRegistry" address="zookeeper://${zookeeper.address2}" />

<dubbo:service interface="org.apache.dubbo.samples.multi.registry.api.HelloService" ref="helloService"
        registry="shanghaiRegistry,beijingRegistry"/>
<dubbo:service interface="org.apache.dubbo.samples.multi.registry.api.DemoService" ref="demoService"
        registry="shanghaiRegistry,beijingRegistry"/>
```

2. 消费端，多订阅

```
<dubbo:registry id="beijingRegistry" address="zookeeper://${zookeeper.address1} " default="false” preferred=“true”
weight=“100”/>
<dubbo:registry id="shanghaiRegistry" address="zookeeper://${zookeeper.address2}” default=“true” weight=“20”/>

<dubbo:reference interface="org.apache.dubbo.samples.multi.registry.api.DemoService"/>
<dubbo:reference  interface="org.apache.dubbo.samples.multi.registry.api.DemoService” registry="beijingRegistry,
shanghaiRegistry"/>
<dubbo:reference interface="org.apache.dubbo.samples.multi.registry.api.HelloService” registry="beijingRegistry"/>
<dubbo:reference interface="org.apache.dubbo.samples.multi.registry.api.HelloService” registry="shanghaiRegistry,
shanghaiRegistry"/>
```

多注册中心

流量调度策略

指定优先级

```
<dubbo:registry  address="zookeeper://${zookeeper.address1} " preferred="true"/>
```

同 Zone 优先

```
<dubbo:registry  address="zookeeper://${zookeeper.address1} " zone="beijing"/>
```

权重轮询

```
<dubbo:registry  address="zookeeper://${zookeeper.address1} " weight="100"/>
```

Stick to 任意可用

多注册中心

适用场景

同区域流量优先调度

出于容灾或者服务伸缩性需求，服务/应用往往需要部署在多个独立的机房/区域，在每个区域有独立注册中心集群的场景下，实现同区域的流量优先调度就能很好的解决延迟和可用性问题。

注册中心迁移

公司的服务一直以来可能是存储在某一个注册中心，如 Zookeeper，但到了某个时间节点，因为各种各样的原因，当我们要迁移到另外的注册中心时，多注册中心模型能够保证平滑的迁移。

异构系统互通

不同微服务体系开发的服务，都封闭在各自的服务发现体系中，而通过统一的多注册中心模型，可以实现不同体系的服务互相发现。

联通 Dubbo 之外的体系

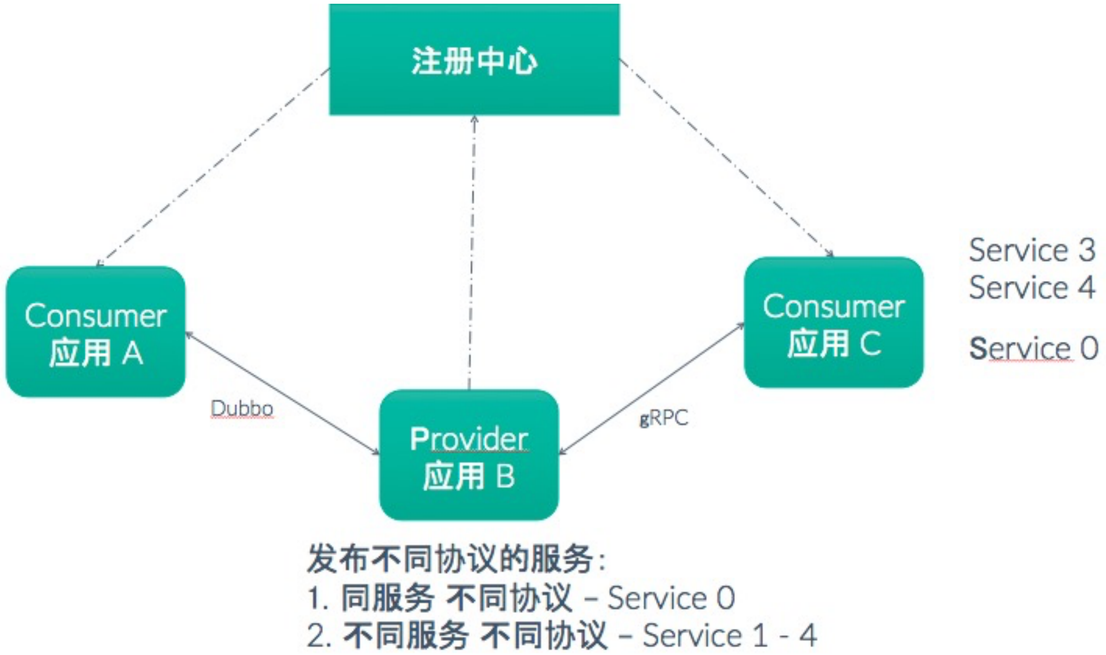
目标

Spring Cloud
gRPC
自建体系

如何交互？

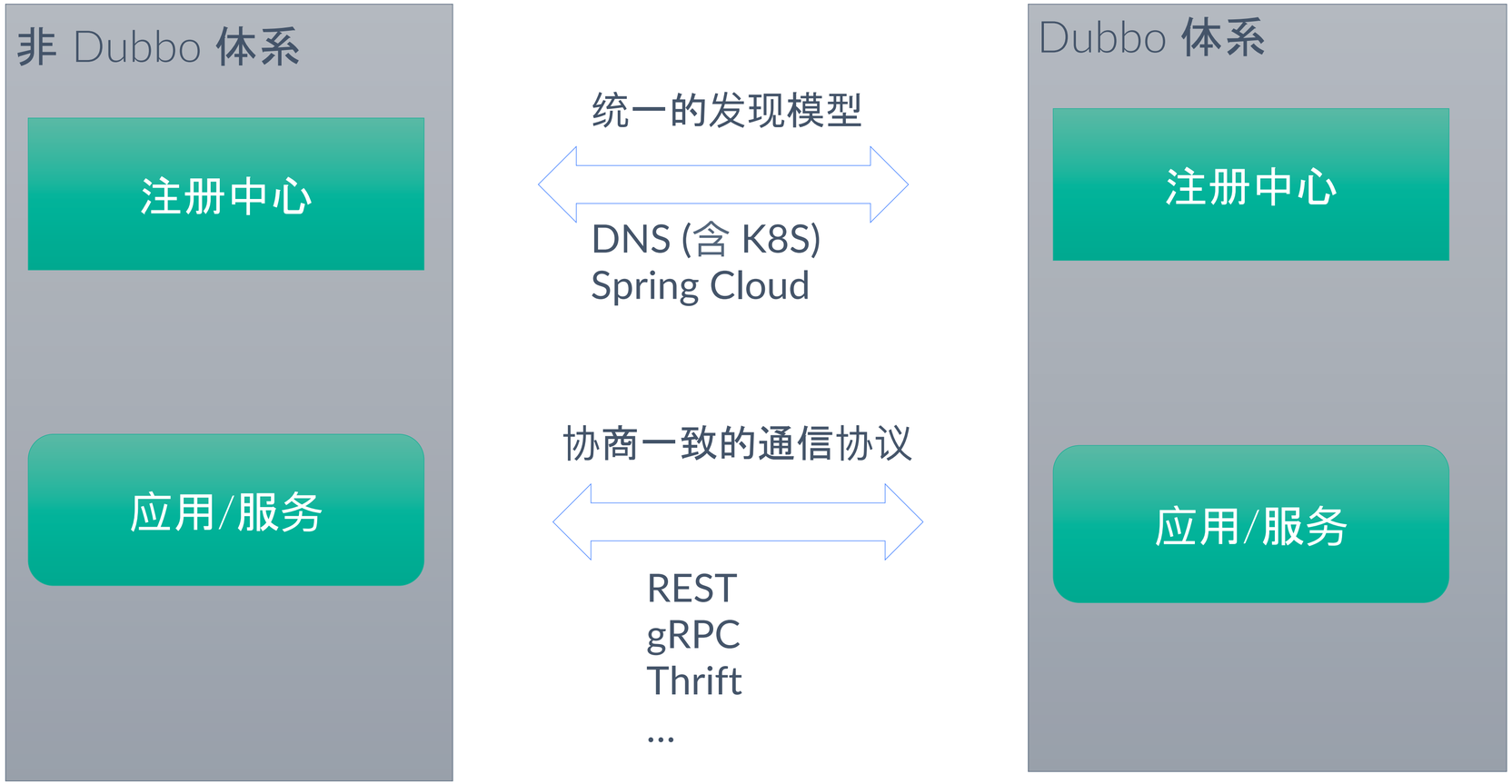
Service 1
Service 2

Service 0



1. 协议打通，两边能以互相理解的协议通信
2. 互相发现，Dubbo 能兼容异构的服务发现体系

联通 Dubbo 之外的体系



如何解决协议互调?

Dubbo

内置高性能私有协议

REST

通用性极强，微服务体系使用最多的协议

gRPC

兼具性能与通用性，同时带来 Stream 传输模型

Thrift

跨语言，是很多企业当前内部 RPC 通信协议

Hessian

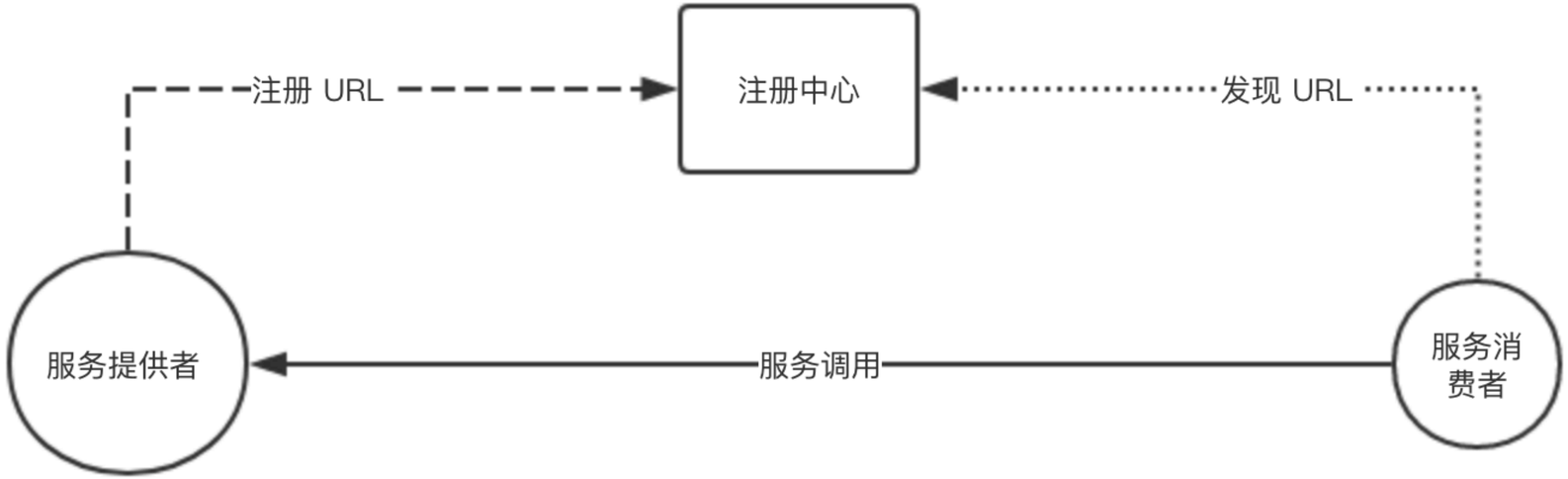
基于 HTTP 的高效二进制通信协议

更多可扩展...

高度可扩展，可随时扩展主流、私有定制协议

如何解决服务发现模型问题?

Dubbo 现有服务发现模型



性能挑战

- 注册中心数据条目和接口数成正比；
- 大量配置信息和地址混合；
- 应用上下线造成地址推送、计算挑战

微服务体系互通

- 和主流微服务体系概念难以对齐
- 无法复用K8S服务基础设施
- 很难对接一些专业的微服务产品，如Consul等

服务治理

- 缺乏应用/实例粒度的服务治理，包括查询和规则下发等；

2. 应用粒度服务注册：服务自省

服务自省

微服务服务发现模型

以应用粒度注册、注册中心只关注地址变更

- IP + PORT；
- 少量实例环境相关配置，如协议、实例所属区域、部署环境等；

元数据服务提供额外信息

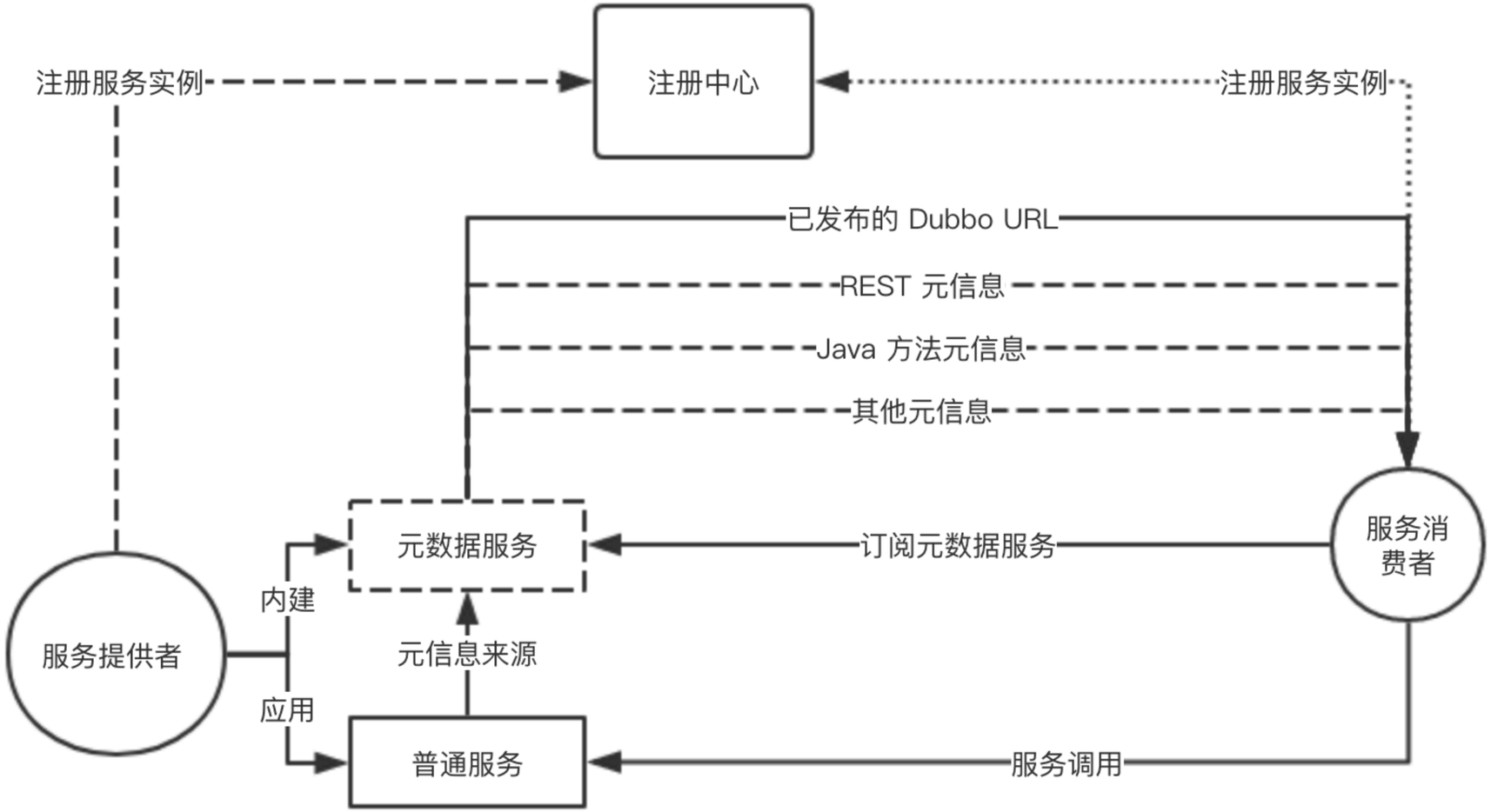
- 接口列表、方法列表、方法签名等；
- 实例特有配置；

保持RPC编程风格不变

- 继续面向接口编程，无需额外改造
- 仅通过Registry配置，区别新老服务发现模型；

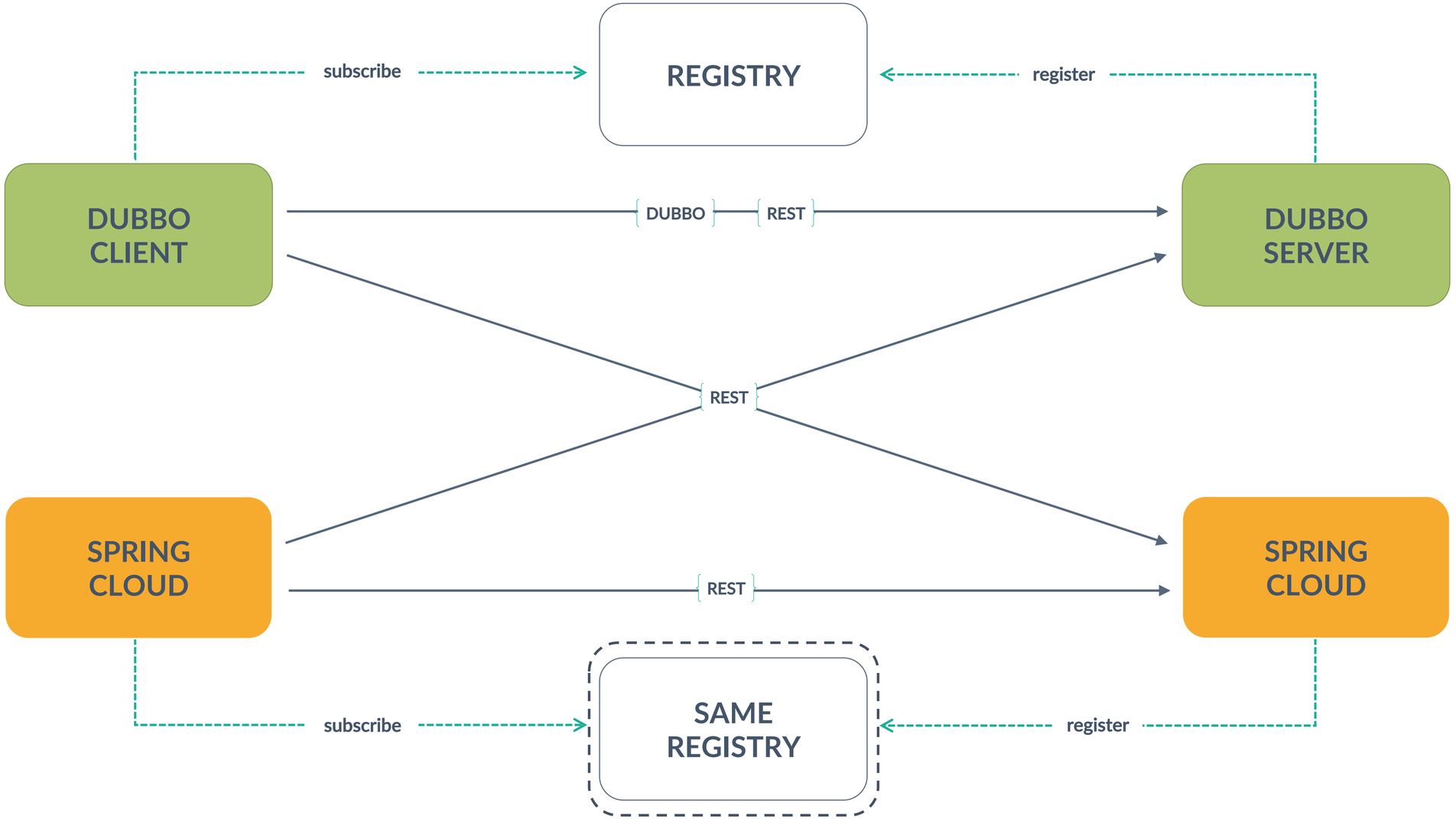
服务自省

微服务服务发现模型



服务自省

与Spring Cloud体系互通



3. 服务自省 示例演示

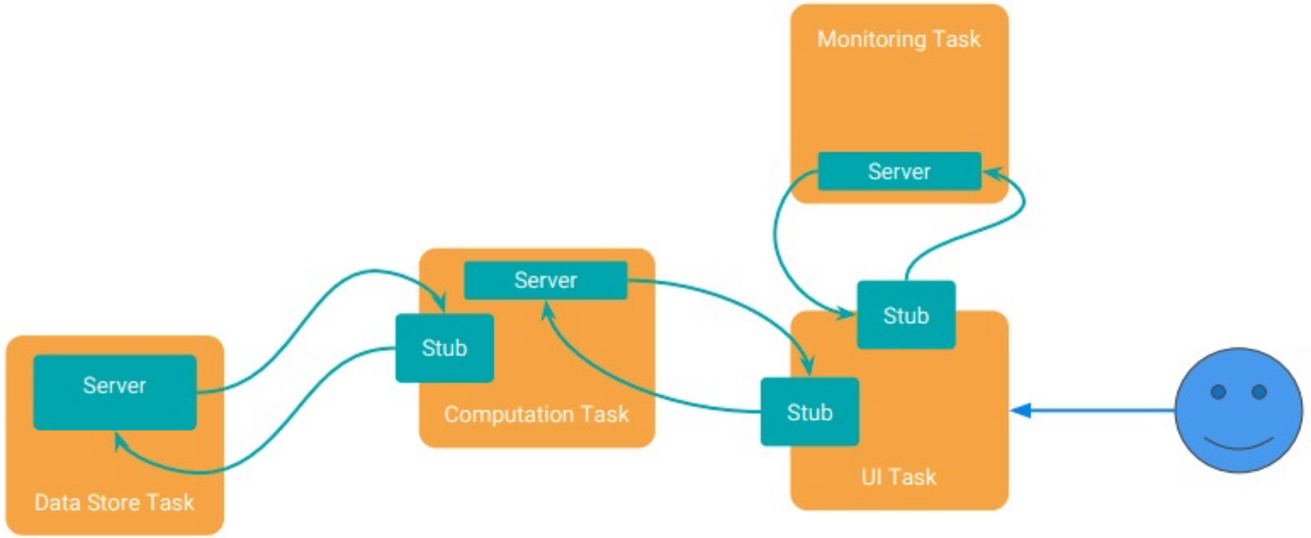
4. 协议支持，HTTP/2 (gRPC)

引言

协议

RPC 对通信要求

- Request - Response
- 链接多路复用
- 双向 Stream 通信模型
- 高效紧凑的应用层协议
- 高性能的序列化协议
- 可扩展
- 普世通用，容易被各层设备识别
- 在性能和通用性间平衡



引言

多语言

多语言

- 多语言是微服务特点
- 场景：技术栈迁移
- RPC 层面要解决的问题
    - 传输层、应用层协议
    - 序列化
    - 服务定义
    - SDK
    - 服务治理

协议

Dubbo 协议

Dubbo Protocol

Offsets Octet		0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Magic High								Magic Low								Req/Res	2 Way	Event	Serialization ID					Status							
4	32	RPC Request ID																															
8	64																																
12	96	Data Length																															
16 ...	128 ...	Variable length part, in turn, is: dubbo version, service name, service version, method name, parameter types, arguments, attachments																															

协议

为什么选择 HTTP 传输层协议

为什么要用 HTTP

- 通用性，连接前后端基础设施
- HTTP 语义和良好的扩展性可满足需求

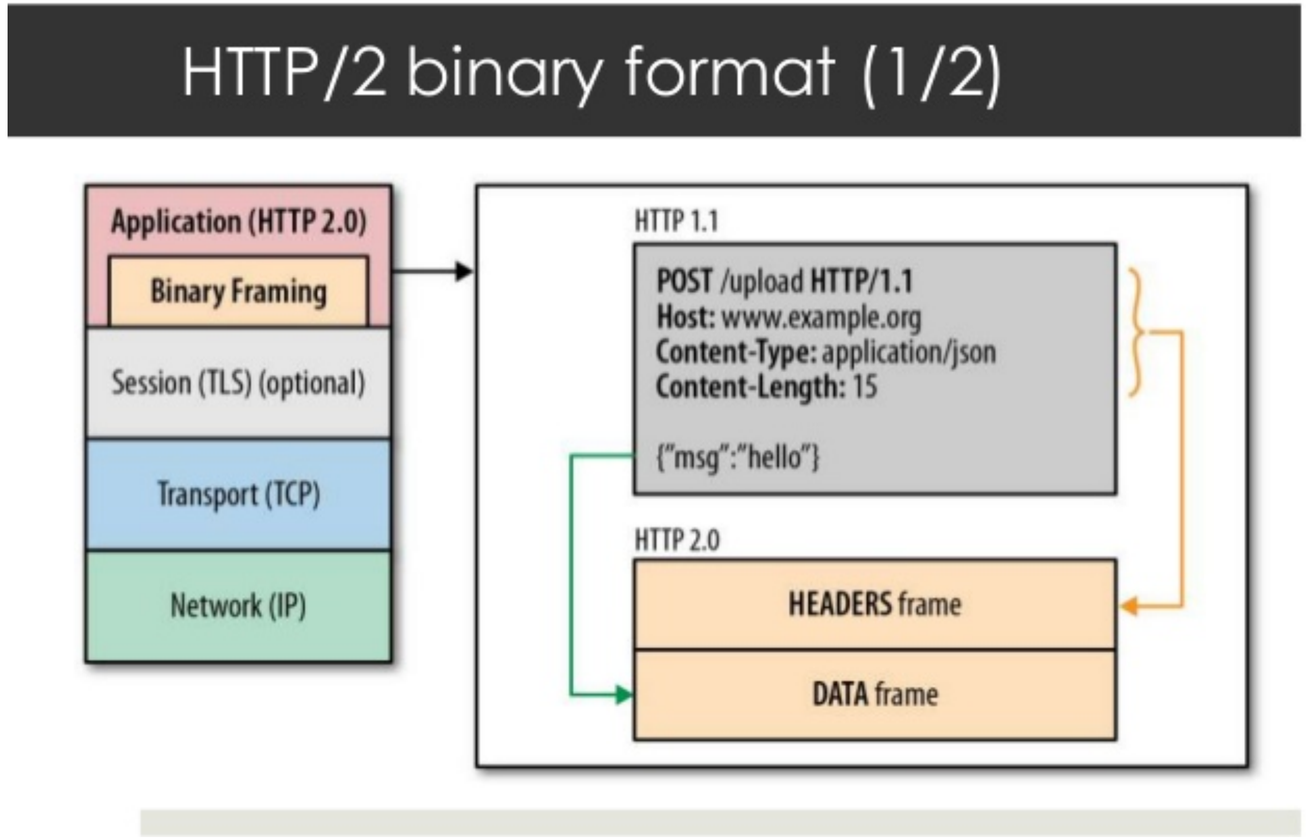
HTTP/1

- Request – Response
- 短链接；Keep-Alive 连接池，仍需要建立多个连接
- Human Readable Headers
- Chunked
- Server Stream

协议

HTTP/2

- Multiplexing，单条链接 based on Frame
- Request - Stream 语义 Server Push
- Flow Control
- 头部压缩 HPACK
- Binary Frame
- TLS



gRPC

- Coverage & Simplicity
- Interoperability & Reach
- General Purpose & Performant
- Payload Agnostic
- Streaming
- Flow Control
- Metadata Exchange

https://platformlab.stanford.edu/Seminar%20Talks/gRPC.pdf

https://grpc.io/blog/principles/?spm=ata.13261165.0.0.2be55017XbUhs8

- 跨平台 跨语言
- 传输层 HTTP/2 + TLS
- 通用 高性能

Protobuf

- 跨语言 跨平台
- 安全性
- 二进制 高性能
- 强类型
- 字段变更向后兼容

- 可扩展，通过扩展 RPC 相关，实现 IDL 服务定义支持

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}
```

Dubbo 做的支持

通过集成 gRPC，使 Dubbo 具备 HTTP/2 通信能力，同时有利于和 gRPC 协议的兼容互调

原生 Dubbo 支持通过 IDL 定义服务，实现跨语言的服务定义

更好的支持语言中立的数据传输

5. gRPC协议 示例演示

Thank You

dubbo.apache.org

Home

dubbo.apache.org

GitHub

github.com/apache/dubbo

Mailing List

dev@dubbo.apache.org

IM

钉钉群：21973601