API设计与实现

主讲人：陈长兵

目 录

CONTENTS

6 API实现-基于RPC

- RPC基础
- RPC框架
- Thrift API实战
- gRPC API实战
- Dubbo API实战

6.1RPC基础

为什么会出现RPC

- 计算机处理器的发展
  - 单核单处理器
  - 多核单处理器
  - 多处理器

6.1RPC基础

为什么会出现RPC

- IPC进程间通信
  - Inter-Process Communication
  - 多CPU之间的数据交换
  - 管道，匿名管道，信号
  - 消息队列，信号量，共享内存区
  - Socket套接字，不同主机上的进程通信

6.1 RPC基础

什么是RPC

- RPC远程过程调用
  - Remote Procedure Call
  - 调用远程方法像调用本地方法一样简单
  - 不同主机上的进程通信
  - 基于Socket套接字

6.1RPC基础

计算机通信本质

- 发送方发送数据，经过一系列传输到接收方

6.1 RPC基础

IPC本质

- I/O操作
  - 磁盘I/O
  - 网络I/O

6.1RPC基础

网络通信

- 网络I/O的基础是Socke通信
  - 从内核空间接收主机数据即准备数据
  - 然后再从内核空间复制到用户空间供用户进行读取

6.1RPC基础

I/O模型

- 阻塞&非阻塞
  - 当前接口数据还未准备就绪时，线程是否被阻塞挂起
  - 阻塞：
  - 非阻塞：

6.1RPC基础

I/O模型

- 同步&异步
  - 当前线程是否需要等待方法调用执行完毕
  - 同步：
  - 异步：

6.1RPC基础

I/O模型

- 用户态&内核态
- 用户线程&内核线程
- 用户空间&内核空间

6.1 RPC基础

I/O模型

- I/O操作两个步骤
  - 发起I/O请求
  - 实际I/O读写（即数据从内核空间到用户空间）

6.1 RPC基础

I/O模型

- 阻塞I/O&非阻塞I/O
  - 指在发起I/O请求时，用户线程是否被阻塞
  - 阻塞I/O：用户线程发起I/O请求的时候，如果数据还未准备就绪，就会阻塞当前线程，让出CPU
  - 非阻塞I/O：用户线程发起I/O请求的时候，如果数据还未准备就绪，不会阻塞当前线程，可以继续执行后面的任务

6.1 RPC基础

I/O模型

- 同步I/O&异步I/O
  - 根据I/O响应方式不同而划分的
  - 同步I/O：用户线程发起I/O请求时，数据是有的，那么将进行实际I/O读写（即数据从内核空间拷贝到用户空间），这个过程用户线程是要等待拷贝完成
  - 异步I/O：用户线程发起I/O请求时，数据是有的，那么将进行实际I/O读写（即数据从内核空间拷贝到用户空间），拷贝的过程不需要用户线程等待，用户线程可以去执行其他逻辑，等内核将数据从内核空间拷贝到用户空间后，用户线程会得到一个“通知”

6.1 RPC基础

Linux I/O模型

- Linux五种I/O模型
  - 阻塞IO，BIO，Blocking IO
  - 非阻塞IO，NIO，None Blocking IO
  - 多路复用IO，IO Multiplexing
  - 信号驱动IO，Signal-driven IO
  - 异步IO，AIO，Asynchronous IO

6.1 RPC基础

Linux I/O模型

- 阻塞IO
  - 默认情况下，所有的socket都是阻塞的
  - 用户进程执行一个系统调用recvfrom，在等待数据到处理数据的两个阶段，整个进程都是被阻塞的

6.1 RPC基础

Linux I/O模型

- 非阻塞IO
  - 通过设置socket变为non-blocking
  - 用户进程执行一个系统调用recvfrom后，进程并没有被阻塞，内核马上返回给进程，如果数据还没准备好，此时会返回一个error
  - 轮询调用recvfrom，直到数据准备好，再拷贝数据到进程，进行数据处理

6.1 RPC基础

Linux I/O模型

- 多路复用IO
  - 通过系统调用select、poll、epoll等，实现对多个IO的监听
  - 当其中任一个socket的数据准备好了，就能返回进行可读
  - 然后用户进程发起recvfrom系统调用，将数据由内核拷贝到用户进程

6.1 RPC基础

Linux I/O模型

- 信号驱动IO
  - 通过系统调用sigaction，配置一个信号处理函数，用户进程继续运行并不阻塞
  - 当数据准备好时，进程会收到一个SIGIO信号
  - 然后可以在信号处理函数中调用I/O操作处理数据

6.1 RPC基础

Linux I/O模型

- 异步IO
  - 用户进程进行aio_read系统调用，无论内核数据是否准备好，都会直接返回给用户进程，然后用户进程可以继续其他任务
  - 等socket数据准备好后，内核直接复制数据给进程
  - 然后从内核向用户进程发通知
  - Linux原生AIO，Linux全新异步I/O框架io_uring

6.1RPC基础

Java I/O模型

- Java三种I/O模型
  - BIO，阻塞IO
  - NIO，New IO
  - AIO，异步IO

6.1 RPC基础

Java I/O模型

- Java BIO
  - 应用程序发起read调用后，会一直阻塞，直到内核把数据拷贝到用户空间

6.1 RPC基础

Java I/O模型

- Java NIO
  - Java1.4引入，在java.nio包
  - 提供Channel、Selector、Buffer等抽象
  - 支持面向缓冲的，基于通道的I/O操作方法

6.1 RPC基础

Java I/O模型

- Java AIO
  - Java7引入，也在java.nio包，也叫NIO2.0
  - 基于事件和回调机制实现的
  - AsynchronousServerSocketChannel，异步操作TCP通道
  - future方式，callback回调方式

6.1 RPC基础

Reactor模式

- 非阻塞同步网络模式
- 基于I/O多路复用机制监听事件，收到事件后，根据事件类型分配给某个进程/线程
- 由Reactor和处理资源池两个核心部分构成
  - Reactor负责监听和分发事件，事件类型包括连接事件、读写事件
  - 处理资源池负责处理事件，如read -> 业务逻辑 -> send

6.1 RPC基础

Reactor模式

- 三种常用类型
  - 单Reactor单进程/线程
  - 单Reactor多进程/线程
  - 多Reactor多进程/线程
- 具体使用进程或线程，与编程语言以及平台有关
  - Java语言一般使用线程，比如netty
  - C语言使用进程和线程都可以，比如Nginx使用进程，Memcache使用的是线程

6.1RPC基础

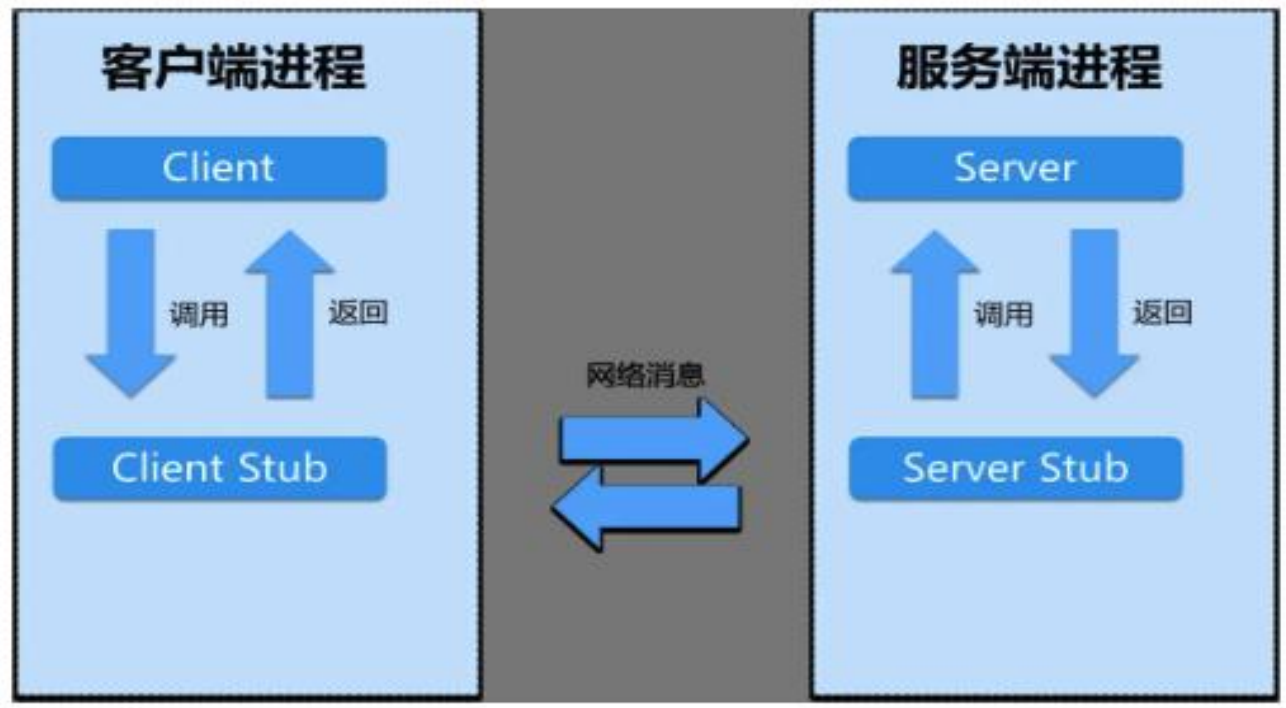
Proactor模式

- 异步网络模式
- 采用异步I/O技术，感知已完成的读写事件
- Linux异步I/O不完善，高性能网络程序都是使用Reactor方案
- Windows提供支持socket的异步编程接口IOCP，

6.1RPC基础

RPC调用过程

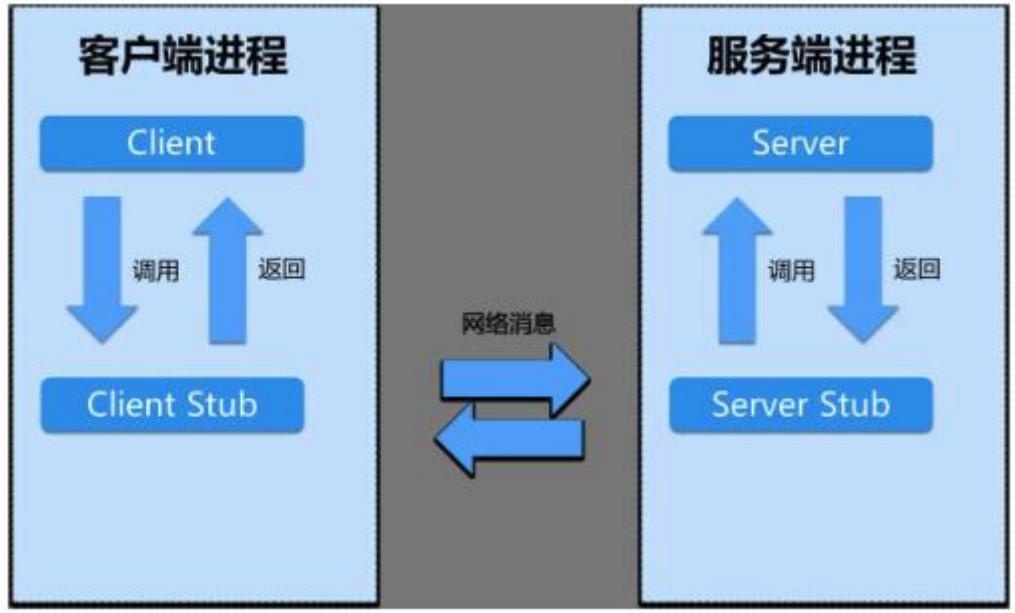
◆RPC由四个部分构成



6.1RPC基础

RPC构成

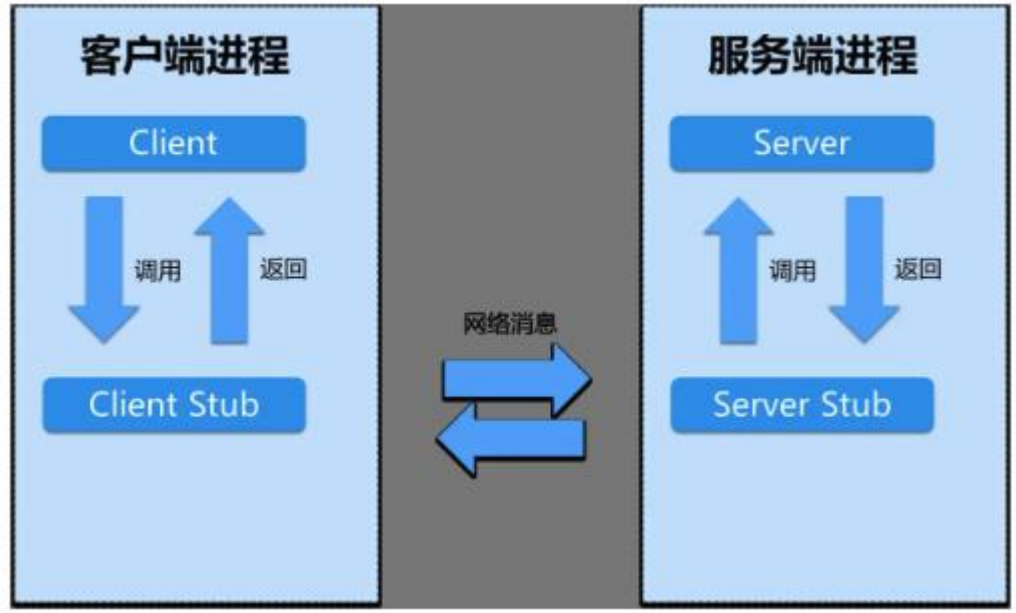
- RPC由四个部分构成
  - 客户端：服务的调用方
  - 客户端存根：client stub，存放服务端的地址信息，再将客户端的请求参数打包成网络消息，然后通过网络远程发送给服务方



6.1RPC基础

RPC构成

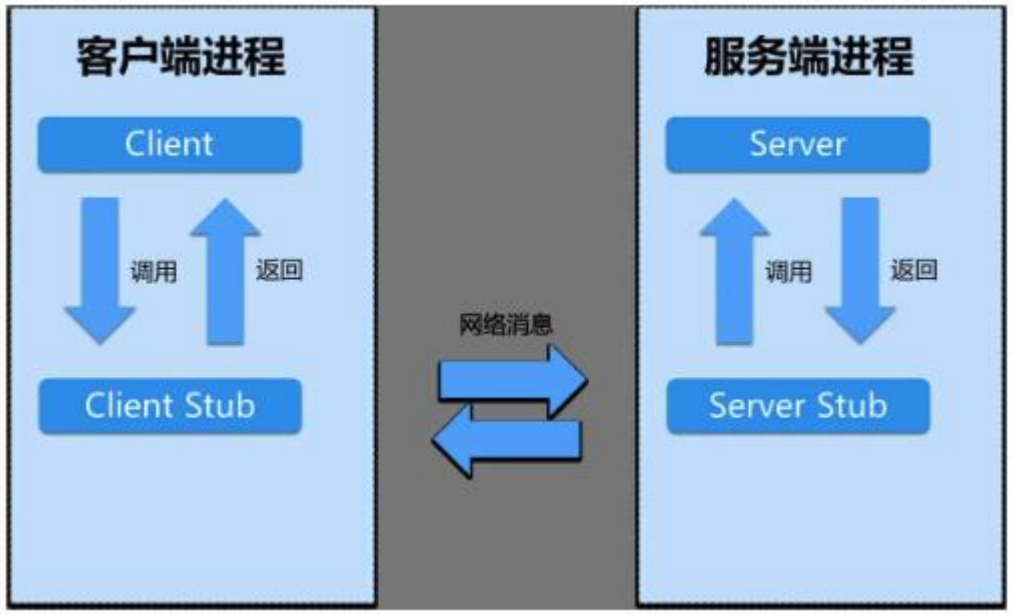
- RPC由四个部分构成
  - 服务端：服务的提供者
  - 服务端存根：server stub，接收客户端发送过来的消息，将消息解包，并调用本地的方法



6.1RPC基础

RPC构成

- 两个概念
  - Marshalling：对请求参数进行封包
  - UnMarshalling：对封包消息进行解包



6.1 RPC基础

RPC 与 RMI

- RPC
  - 远程过程调用
  - Remote Procedure Call
  - 面向过程
- RMI
  - 远程方法调用
  - Remote Method Invocation
  - 面向对象

6.1 RPC基础

RPC 与 RMI

- 两个概念
  - stub，存根，客户端代理
  - skeleton，骨架，服务端代理

6.2RPC框架

为什么需要RPC框架

- 网络框架
  - 解决网络通信的某一类问题
  - 框架与类库的区别，框架内部的类有相互协作的功能
  - Netty框架，Socket/NIO
- RPC框架
  - 不用感知RPC调用实现的细节

6.2RPC框架

RPC框架选型要素

- 语言和平台中立
  - 拆分模块未必使同同一种语言开发，或部署在不同的平台

6.2RPC框架

RPC框架选型要素

- 服务治理相关指标
  - 服务注册、路由策略、负载策略等

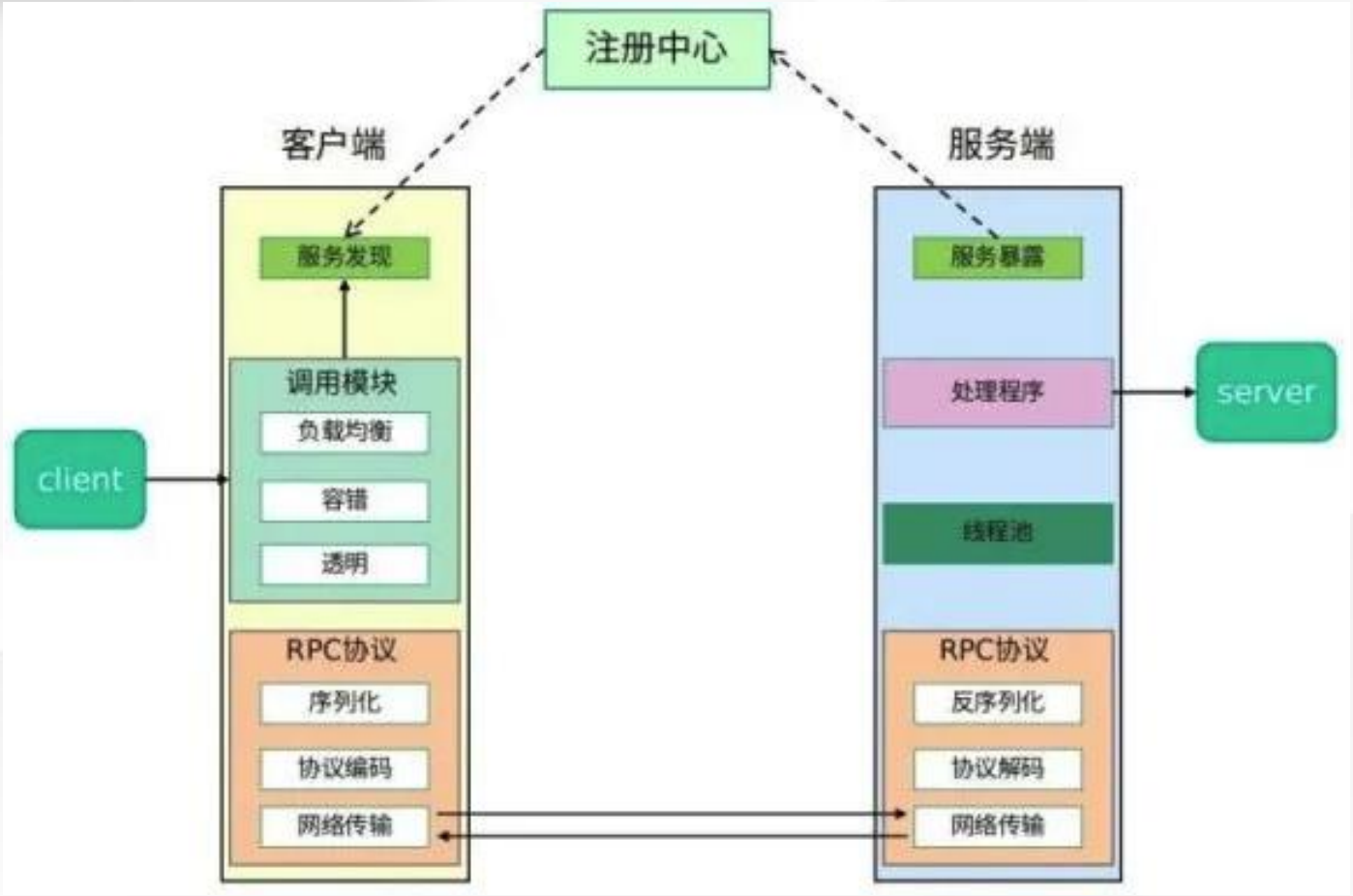
6.2RPC框架

RPC框架选型要素

- 可靠性和稳定性
- 性能
- 设计
- 文档
- 开源社区的成熟度

6.2RPC框架

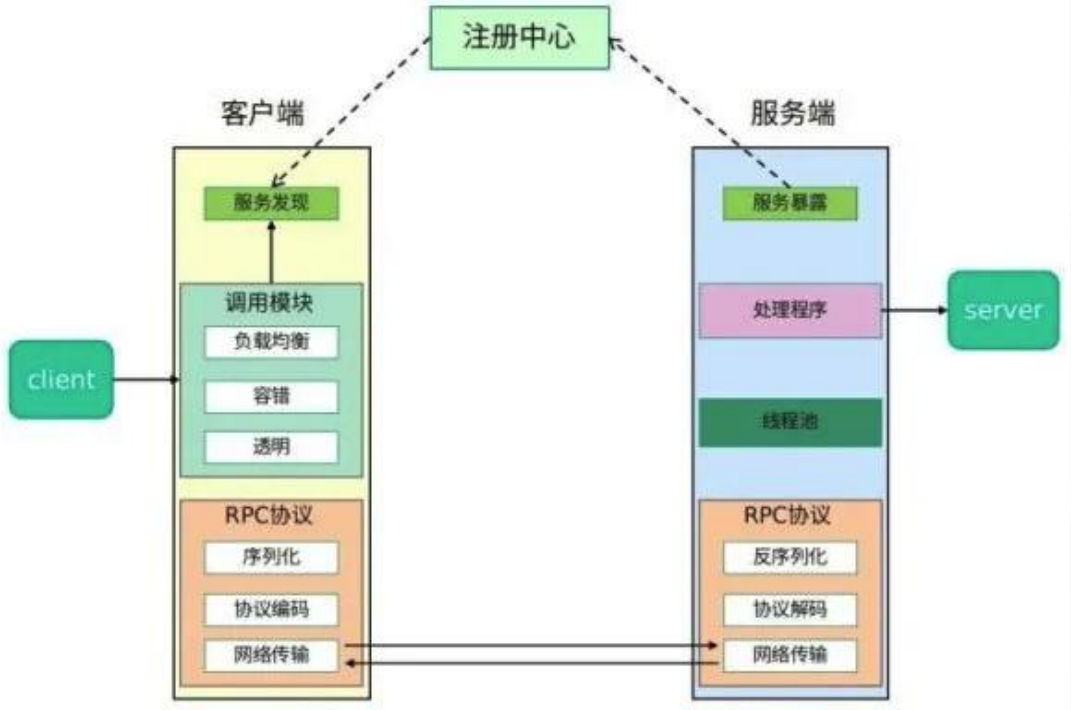
RPC框架核心功能



6.2RPC框架

RPC框架核心功能

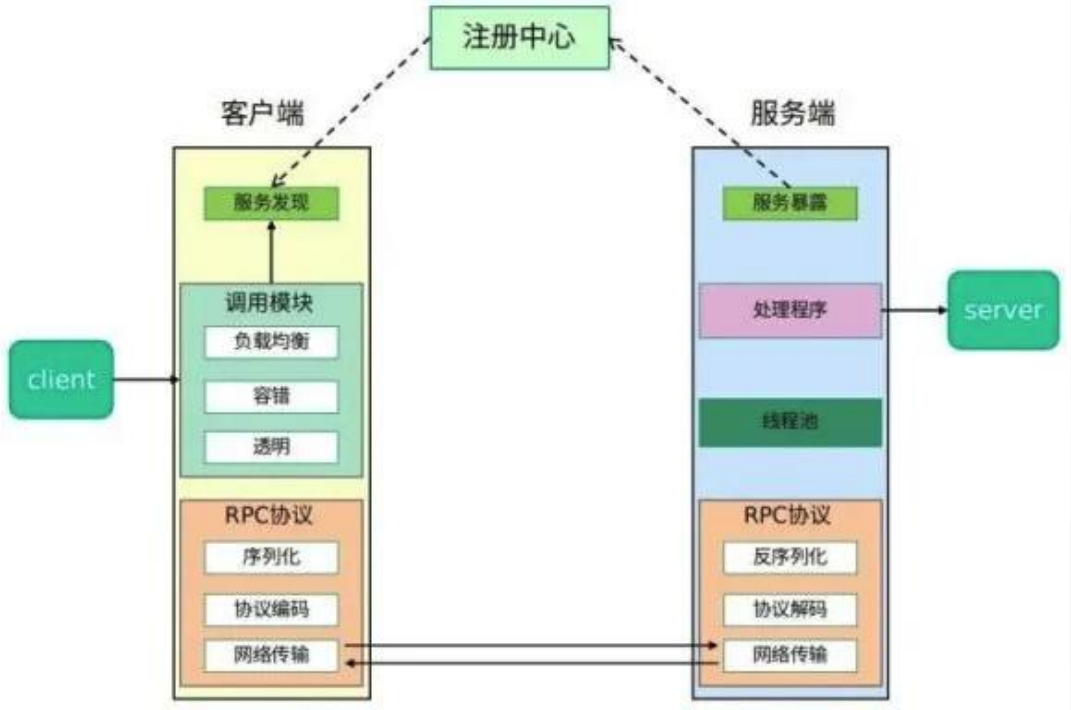
- 服务管理
  - 定义
  - 注册
  - 发现



6.2RPC框架

RPC框架核心功能

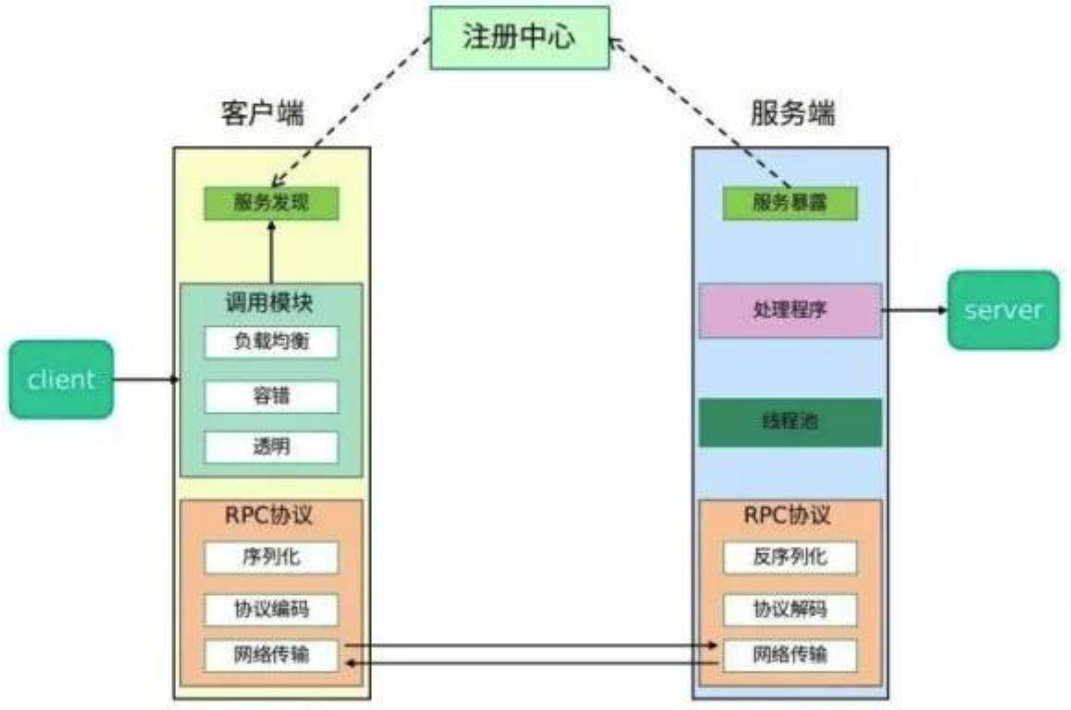
- RPC调用
  - 负载均衡
  - 容错
  - 透明



6.2RPC框架

RPC框架核心功能

- RPC协议
  - 序列化
  - 协议编码
  - 网络传输



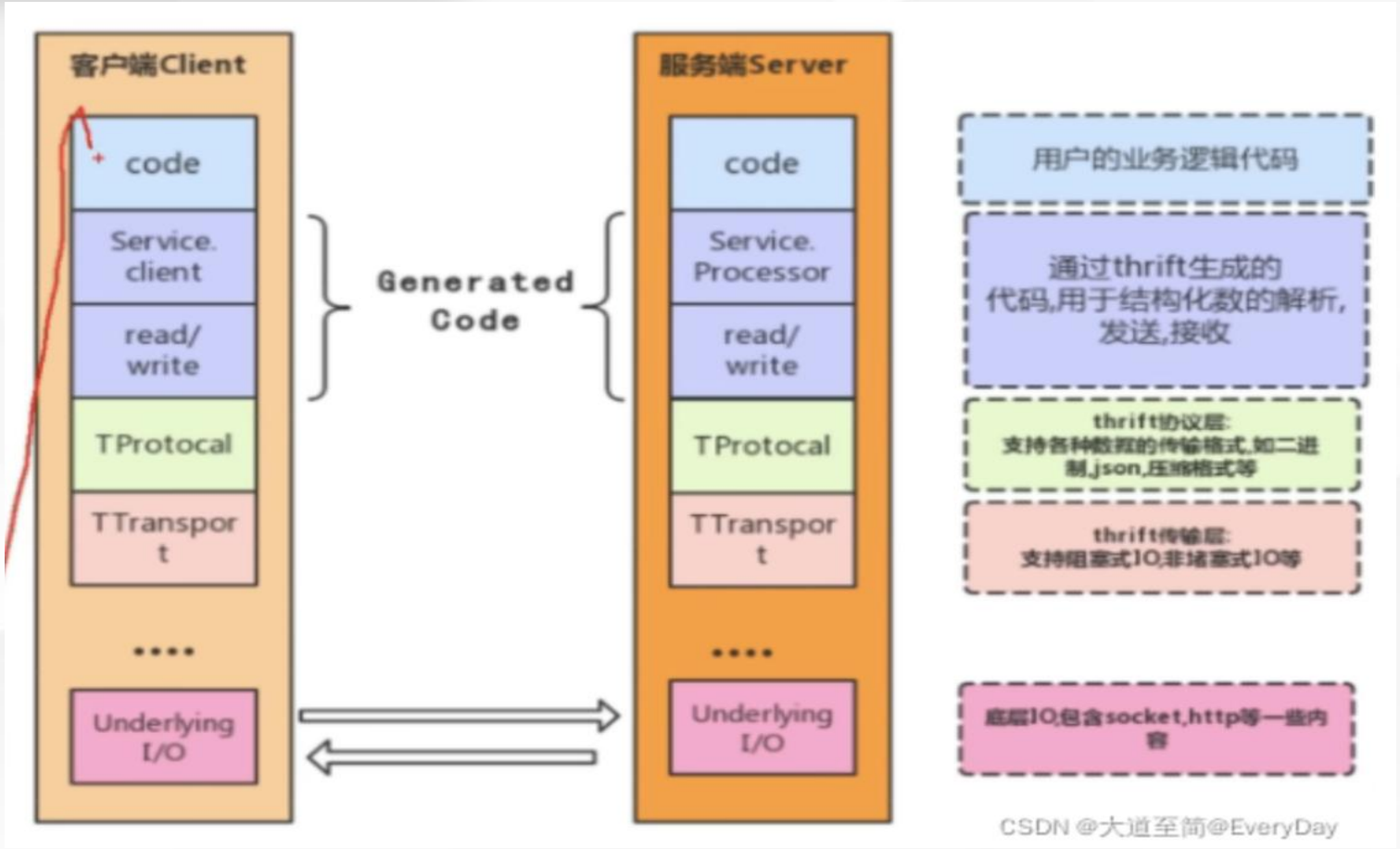
6.2RPC框架

常见RPC框架

- Thrift
  - 由Facebook贡献给Apache开源项目
  - 基于静态代码生成的跨语言RPC框架
- gRPC
  - 由Google开发的高性能、通用的开源RPC框架
  - 简单服务定义，跨平台跨语言，基于http/2双向流传输
- Dubbo
  - 阿里开源的基于Java的RPC分布式服务框架
  - 提供高性能和透明化的RPC远程服务调用，以及SOA服务治理方案

6.3 Thrift API实战

Thrift架构



6.3 Thrift API实战

Thrift架构

- TTransport
  - 实现RPC的传输层，定义具体的网络传输协议
- TProtocol
  - 实现RPC的协议层，定义数据传输格式，负载网络传输数据的序列化
- TProcessor
  - 作为协议层和用户提供的服务实现之间的纽带，负责调用服务实现的接口
- TServer
  - 整合TProtocol，TTransport和TProcessor几个对象

6.3 Thrift API实战

Thrift协议

- 分为文本和二进制传输协议
  - 一般采用二进制类型
- 常用协议
  - TBinaryProtocol
  - TCompactProtocol
  - TJSONProtocol
  - TSimpleJSONProtocol

6.3 Thrift API实战

Thrif传输层

- 常用传输层对象
  - TIOStreamTransport
  - TSocket
  - TNonblockingTransport
  - TNonblockingSocket
  - TMemoryBuffer
  - TMemoryInputTransport
  - TFrameTransport

6.3 Thrift API实战

Thrif网络服务模型

- 阻塞服务模型
  - TSimpleServer
  - TThreadPoolServer
- 非阻塞服务模型
  - TNonblockingServer
  - THashServer
  - TThreadedSelectorServer

6.3 Thrift API实战

Thrif IDL介绍

- IDL，interface description language
  - 定义RPC的数据类型和接口
  - 由Thrift代码生成工具处理，生成各类目标语言的代码
  - 以支持IDL文件中定义的结构和服务实现跨语言服务调用。
  - https://thrift.apache.org/docs/idl.html

6.3 Thrift API实战

Thrif IDL介绍

- 基本类型
  - 不支持服无符号类型
  - byte
  - i16，i32，i64
  - double
  - string

6.3 Thrift API实战

Thrif IDL介绍

- 容器类型
  - 支持除service之外的任何类型，包括exception
  - list<T>，有序列表
  - set<T>，无序集合
  - map<K, V>，字典结构

6.3 Thrift API实战

Thrif IDL介绍

- 结构体
  - 将一些数据聚合在一起，方便传输管理
  - 定义形式

```
struct People {
    1: string name;
    2: i32 age;
    3: string sex;
}
```

6.3 Thrift API实战

Thrif IDL介绍

- 枚举
  - 定义形式

```
enum Sex {
    MALE;
    FEMALE;
}
```

6.3 Thrift API实战

Thrif IDL介绍

- 异常类型
  - 规则与struct一样，定义形式

```
exception RequestException {
    1: i32 code;
    2: string reason;
}
```

6.3 Thrift API实战

Thrif IDL介绍

- 服务
  - 定义服务类似创建接口
  - 创建的service经过代码生成命令后就会生成客户端和服务端的框架代码
  - 定义形式如下：

```
service HelloWordService {
    string doAction(1: string name, 2: i32 age);
}
```

6.3 Thrift API实战

Thrif IDL介绍

- 类型定义
  - 类似c++的typedef定义
  - 定义形式如下：

```
typedef i32 Integer
typedef i64 Long
```

6.3 Thrift API实战

Thrif IDL介绍

- 常量
  - 使用const关键字
  - 定义形式如下：

```
const i32 MAX_RETRIES_TIME = 10
const string MY_WEBSITE = "http://demo.com";
```

6.3 Thrift API实战

Thrif IDL介绍

- 命名空间
  - 类似java中的package，主要目的是组织代码
  - 定义形式如下：

```
namespace java com.winwill.thrift
```

6.3 Thrift API实战

Thrif IDL介绍

- 文件包含
  - 相当于C/C++中的include，Java中的import
  - 定义形式如下：

```
include "global.thrift"
```

6.3 Thrift API实战

Thrif IDL介绍

- 注释
  - 注释方式支持shell风格的注释，支持C/C++风格的注释
  - #和//开头的语句都单行做注释，/**/包裹的语句也是注释。

6.3 Thrift API实战

Thrif IDL介绍

- 可选与必选
  - 提供required，optional两个关键字
  - 定义形式

```
struct People {
    1: required string name;
    2: optional i32 age;
}
```

6.3 Thrift API实战

环境准备

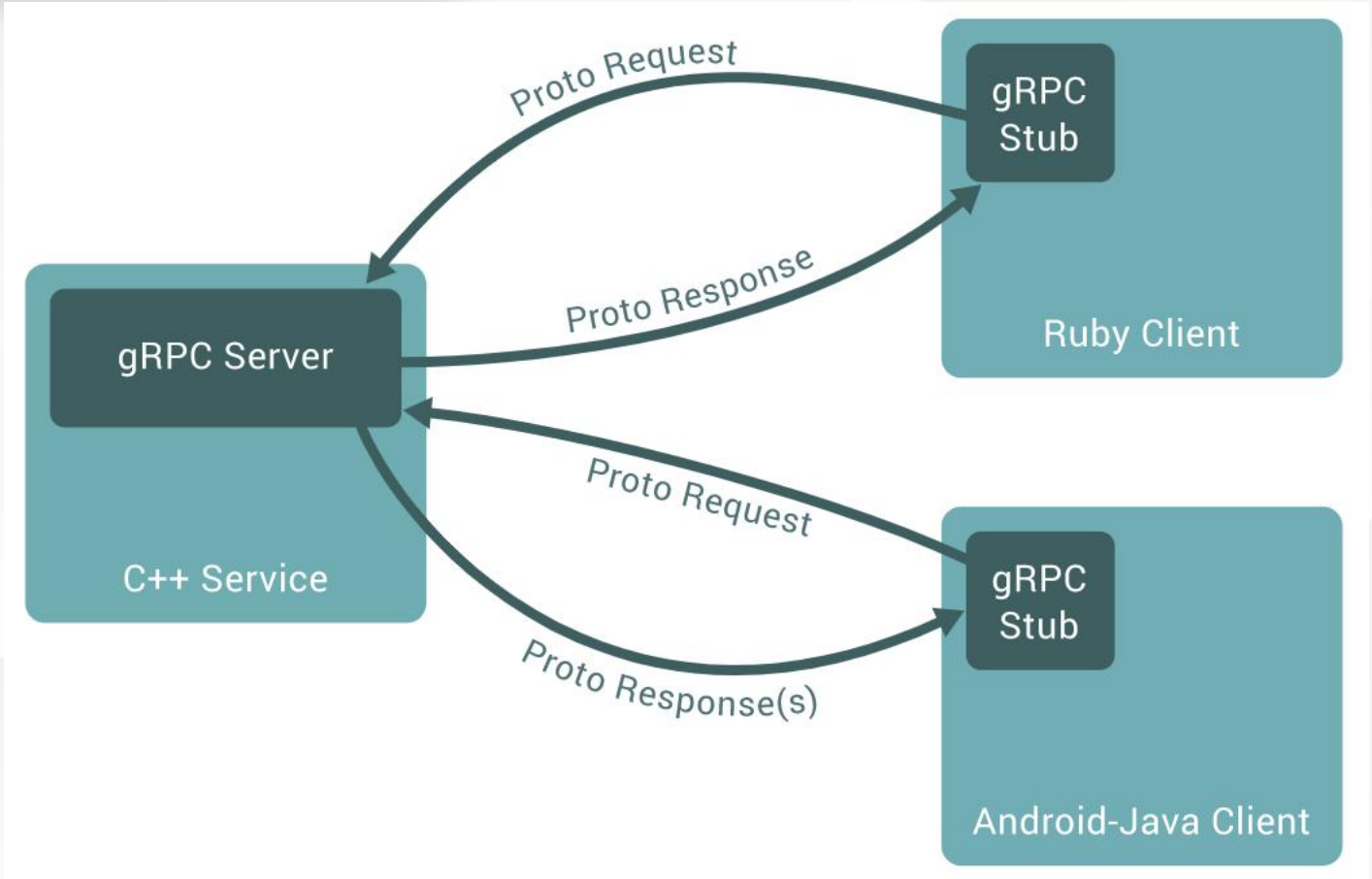
- 下载地址
  - https://thrift.apache.org/download
  - thrift-0.18.0.tar.gz
  - thrift-0.18.0.exe
- 官方示例
  - https://thrift.apache.org/tutorial/

6.3 Thrift API实战

案例分析与实现

6.4 gRPC API实战

gRPC框架



6.4 gRPC API实战

gRPC框架

业务模块的数据，通信双方需要了解彼此的数据模型，才能正确交互信息

gRPC通过Protocol Buffers编码格式承载数据

远程过程调用，定义了远程过程调用的协议交互格式

gRPC承载在HTTP 2.0协议上

TCP连接提供面向连接的、可靠的、顺序的数据链路

6.4 gRPC API实战

HTTP协议

- HTTP1.0
  - 短连接
- HTTP1.1
  - 长链接，丰富Header语言，Head-of-Line Blocking
- HTTP2.0
  - 多路复用，服务端推送

6.4 gRPC API实战

HTTP协议

- HTTP2.0基本概念
  - Connection：一个TCP连接
  - Stream：一个双向流，一个连接可以有多个Stream
  - Message：逻辑上的request，response
  - Frame：数据传输的最小单位。每个Frame都属于一个特点的Stream或整个连接。一个Message可能有多个Frame组成。比如一个请求消息，包括一个HEADERS帧和多个DATA帧。

6.4 gRPC API实战

HTTP协议

- HTTP2.0 Frame格式
  - Length(24)，Frame长度
  - Type(8)，Frame类型，如DATA，HEADERS，PRIORITY等
  - Flag(8) 和 R(1)，保留位
  - Stream Identifier(31)，标识所属的stream，如果为0，则表示属于整个连接。
  - Frame Payload，不同Type有不同的格式

6.4 gRPC API实战

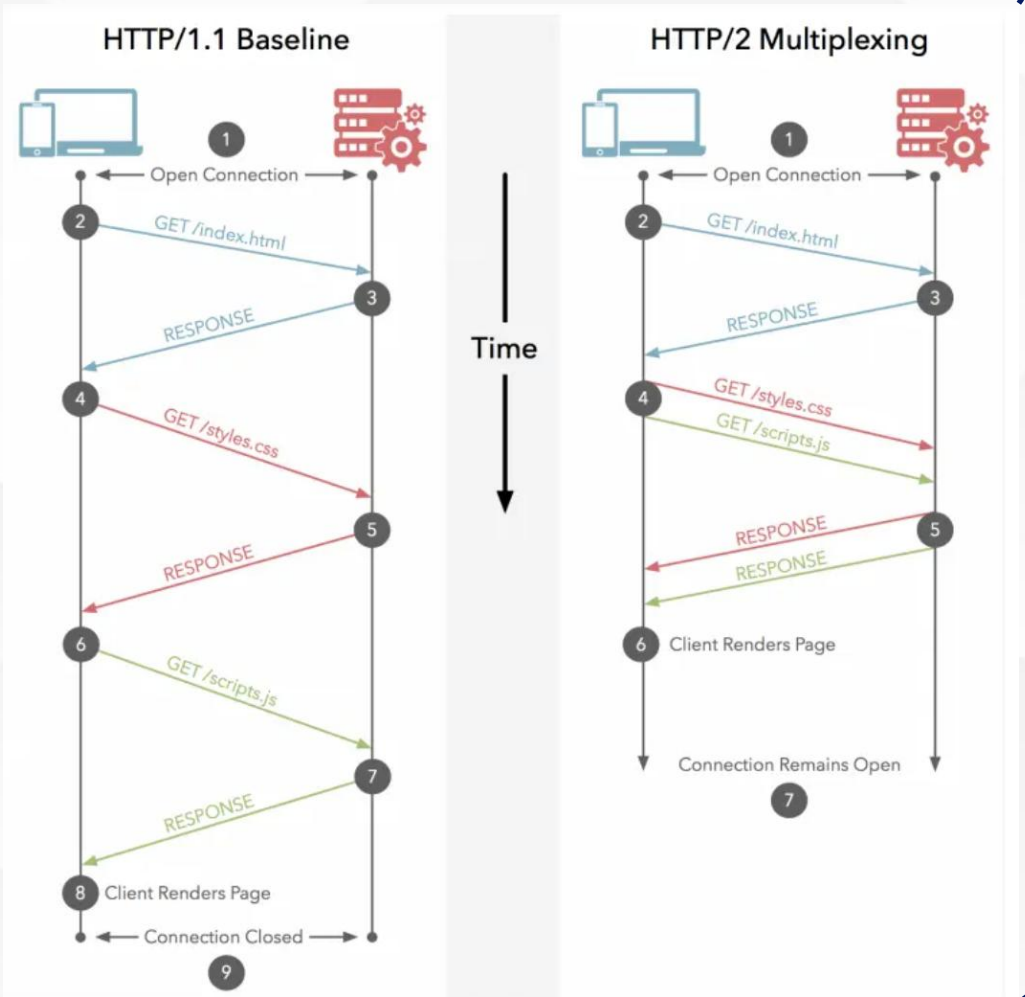
HTTP协议

- HTTP2.0 Stream特性
  - 一条连接可以包含多个Stream，发送数据互不影响
  - Stream可被client和server单方面或共享使用
  - Stream可被任意一端关闭
  - Stream用一个唯一ID来标识
  - 客户端发起的流有奇数ID，服务器端发起的流具有偶数ID
  - 同一个Stream的Frame必须是有序的

6.4 gRPC API实战

HTTP协议

- HTTP2.0 多路复用
  - 一个request对应一个stream
  - 一个连接有多个stream
  - 不同stream的frame可混杂在一起
  - 接收方根据stream id再将frame整合到不同的request中



6.4 gRPC API实战

HTTP协议

- HTTP2.0 服务端推送
  - HTTP1.x，服务端不具备主动推送数据资源给客户端的能力，请求-响应模型
  - HTTP1.x，长轮询，每隔一段时间向服务器发起查询请求
  - WebSocket，双向通信，建立在TCP之上
  - 服务端在客户端已发起的请求流中，先发送一个PUSH_PROMISE类型的Frame，再发送正常的响应
  - 然后服务端发送推送信息，stream id与PUSH_PROMISE类型帧中的Promised Stream ID一致

6.4 gRPC API实战

HTTP协议

- HTTP2.0 其他特性
  - Header压缩
  - 流量控制
  - 更安全的SSL

6.4 gRPC API实战

Protocol Buffers介绍

- 数据表达方式，类似xml、json，以.proto结尾的数据文件
- 使用protoc工具生成各种语言的代码
- 编解码速度快，传输数据更小

6.4 gRPC API实战

Protocol Buffers介绍

- TLV数据存储
  - Tag-Length-Value，标识-长度-字段值
  - Tag：字段标识号，用于标识字段
  - Length：Value的字节长度
  - Value：消息字段经过编码后的值
  - 自描述？

6.4 gRPC API实战

Protocol Buffers介绍

- TLV数据存储
  - Varint编码
  - ZigZag编码
  - length-delimi编码
  - Fixe编码

6.4 gRPC API实战

Protocol Buffers介绍

- 数据类型
  - double，float
  - int32，int64，uint32，uint64，sint32，sint64
  - fixed32，fixed64，sfixed32，sfixed64
  - bool，string，bytes
  - enum
  - message
  - map<key_type, value_type>

6.4 gRPC API实战

Protocol Buffers介绍

- 定义message类型
  - 指定版本，proto2/proto3
  - 指定字段类型
  - 指定字段编号，1-15单字节，16-2047两个字节
  - 指定字段规则，singular，optional，repeated，map

```
syntax = "proto3";

message SearchRequest {
  string query = 1;
  int32 page_number = 2;
  int32 result_per_page = 3;
}
```

6.4 gRPC API实战

Protocol Buffers介绍

- 指定字段规则
  - singular（默认），optional，repeated，map
  - proto3语法没有required，所有字段都是可选的

```
message Person {
  string name = 1;
  int32 age = 2;
  repeated string loction = 3;
}
```

6.4 gRPC API实战

Protocol Buffers介绍

- 定义枚举类型
  - 通常第一个值编码为0
  - 在message使用时，第一个值编码必须为0
  - 在message内部或外部都可定义枚举类型

```
enum Corpus {
  CORPUS_UNSPECIFIED = 0;
  CORPUS_UNIVERSAL = 1;
  CORPUS_WEB = 2;
  CORPUS_IMAGES = 3;
  CORPUS_LOCAL = 4;
  CORPUS_NEWS = 5;
  CORPUS_PRODUCTS = 6;
  CORPUS_VIDEO = 7;
}

message SearchRequest {
  string query = 1;
  int32 page_number = 2;
  int32 result_per_page = 3;
  Corpus corpus = 4;
}
```

6.4 gRPC API实战

Protocol Buffers介绍

- 默认值
  - proto3不再支持default关键字设置默认值
  - strings空串，bytes空字节
  - bool类型false，数值类型0
  - enums第一个定义元素
  - message字段，与特定语言相关
  - 标量类型的默认值不会序列化

6.4 gRPC API实战

Protocol Buffers介绍

- 预留字段
  - proto文件升级，更新或删除字段，防止解析旧版本文件异常
  - 可使用字段名字 或 字段编码任一方式，不能混用

```
message Foo {
  reserved 2, 15, 9 to 11;
  reserved "foo", "bar";
}
```

6.4 gRPC API实战

Protocol Buffers介绍

- 使用其他message
  - 同文件引用
  - 不同文件引用
  - 使用proto2中的message类型，proto2中的enum无法直接使用

```
import "myproject/other_protos.proto";

message SearchResponse {
  repeated Result results = 1;
}
```

6.4 gRPC API实战

Protocol Buffers介绍

- 嵌套类型
  - 父message之外可引用嵌套的message
  - 多层嵌套

```
message SearchResponse {
  message Result {
    string url = 1;
    string title = 2;
    repeated string snippets = 3;
  }
  repeated Result results = 1;
}
```

6.4 gRPC API实战

Protocol Buffers介绍

- 注释
  - 使用c/c++风格，// 或 /* ... */

```
/* SearchRequest represents a search query, with pagination options to
 * indicate which results to include in the response. */

message SearchRequest {
  string query = 1;
  int32 page_number = 2;  // Which page number do we want?
  int32 result_per_page = 3;  // Number of results to return per page.
}
```

6.4 gRPC API实战

Protocol Buffers介绍

- message更新
  - 不要更改已存在字段的编号
  - 可新增field，新/旧代码都可以序列化/反序列化新/旧message
  - 可删除field，但要保证对应字段编号不再被使用
  - int32、uint32、int64、uint64、bool是互相兼容的
  - sint32、sint64是互相兼容的
  - string和bytes是互相兼容的，但要使用utf-8编码
  - 如果byte包含message的编码版本，且嵌套的message和byte兼容
  - fixed32兼容sfixed32，fixed64，sfixed64
  - enum兼容int32，uint32，int64，uint64

6.4 gRPC API实战

Protocol Buffers介绍

- Options选项
  - 不会改变声明的整体含义，但会影响在特定上下文中处理声明的方式
  - file option, field option

```
option java_multiple_files = true;
option java_package = "io.grpc.examples.routeguide";
option java_outer_classname = "RouteGuideProto";
option objc_class_prefix = "RTG";
```

6.4 gRPC API实战

Protocol Buffers介绍

- 定义服务

```
service SearchService {
  rpc Search(SearchRequest) returns (SearchResponse);
}
```

6.4 gRPC API实战

gRPC四种通信模式

- 简单RPC
- 服务端流式RPC
- 客户端流式RPC
- 双向流式RPC

6.4 gRPC API实战

gRPC四种通信模式

- 简单RPC
  - 客户端一次请求，服务器一次应答

```
rpc simpleHello(Request) returns (Result) {}
```

6.4 gRPC API实战

gRPC四种通信模式

- 服务端流式RPC
  - 客户端一次请求，服务端多次应答（流式）
  - 客户端发出请求，不一定会立即得到一个相应
  - 在服务端与客户端建立一个单向流，服务端可以随时向流中写入多个响应消息
  - 典型场景：股票实时数据

```
rpc serverStreamHello(Request) returns (stream Result) {}
```

6.4 gRPC API实战

gRPC四种通信模式

- 客户端流式RPC：
  - 客户端多次请求（流式），服务器一次应答
  - 典型场景：物联网终端向服务器报送数据

```
rpc clientStreamHello(stream Request) returns (Result) {}
```

6.4 gRPC API实战

gRPC四种通信模式

- 双向流式RPC：
  - 客户端多次请求（流式），服务器多次应答（流式）
  - 典型场景：聊天应用

```
rpc biStreamHello(stream Request) returns (stream Result) {}
```

6.4 gRPC API实战

gRPC环境准备

- 下载protoc，处理proto文件，生成代码
  - https://repo1.maven.org/maven2/com/google/protobuf/protoc/3.22.0/
- 下载rpc插件，处理rpc定义的插件
  - https://repo1.maven.org/maven2/io/grpc/protoc-gen-grpc-java/
- 下载wrapper.proto
  - https://github.com/protocolbuffers/protobuf/blob/main/src/google/protobuf/wrappers.proto

6.4 gRPC API实战

gRPC案例分析与实现

6.5 Dubbo API实战

Dubbo的演进

- 2011，阿里开源，发布2.0.7-2.0.9
- 2012，发布2.0.10-2.5.3，发布2.5.3后停止更新
- 2013-2014，更新2次2.4的维护版本
- 2017，重启更新，发布2.5.4-2.5.8
- 2018，发布2.6.x版本，进入apache基金孵化器
- 2019，成为apache顶级项目
  - 两个长期维护版本，Dubbo 2.6.x ，Dubbo 2.7.x （apache 孵化版本）
- 2021，发布全新的3.x版本，几乎兼容2.7.x所有行为

6.5 Dubbo API实战

Dubbo是什么

- 微服务编程范式与工具
  - 支持基于 IDL 或语言特定方式的服务定义
  - 提供多种形式的服务调用形式（如同步、异步、流式等）
- 高性能的RPC通信
  - 提供基于 HTTP、HTTP/2、TCP 等多种高性能通信协议实现
  - 并支持序列化协议扩展

6.5 Dubbo API实战

Dubbo是什么

- 微服务监控与治理
  - 官方提供的服务发现、动态配置、负载均衡、流量路由等基础组件
  - 用 Admin 控制台监控微服务状态，通过周边生态完成限流降级、数据一致性、链路追踪等能力

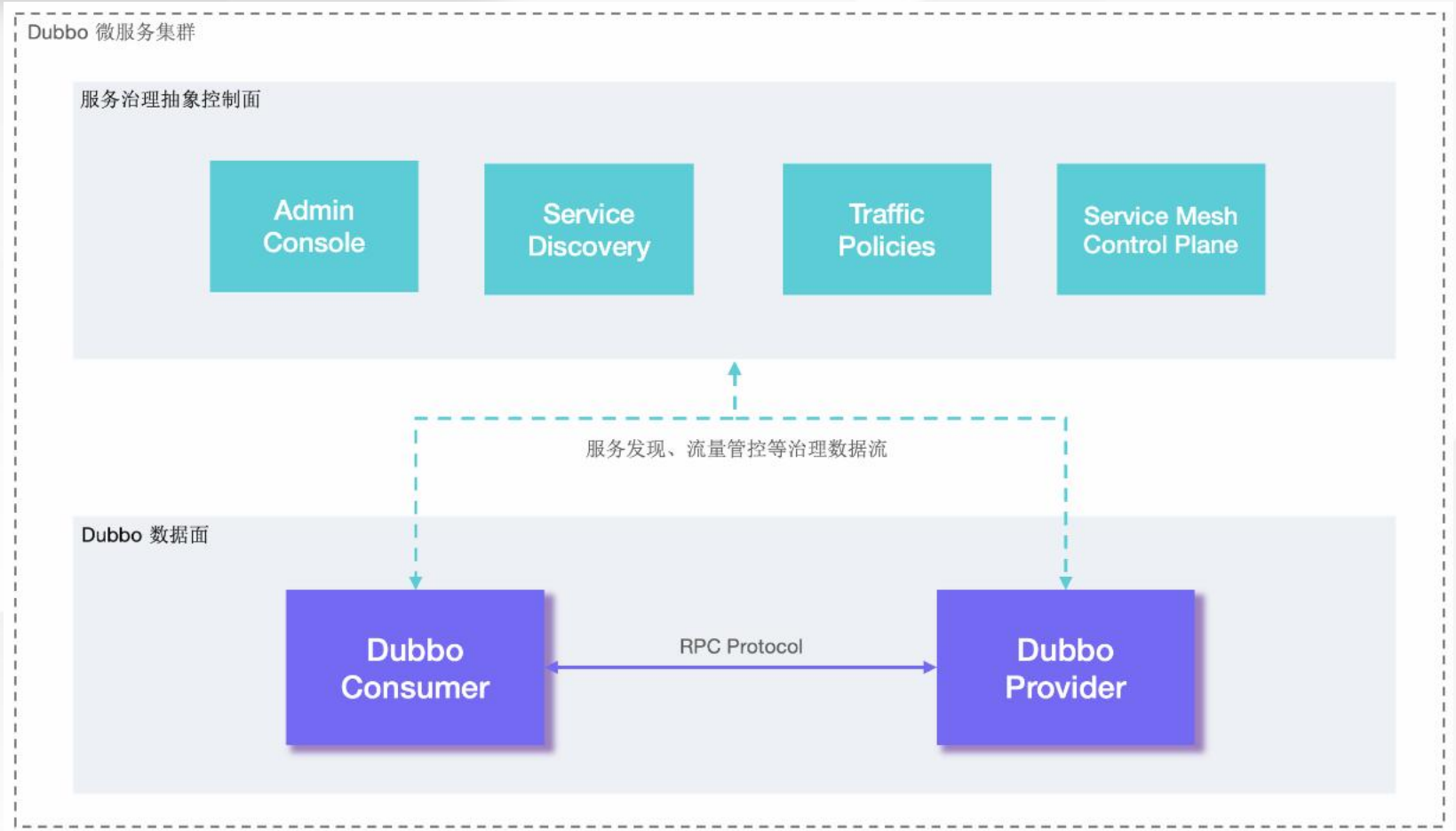
6.5 Dubbo API实战

Dubbo不是什么

- 不是应用开发框架的替代者
  - 设计为让开发者以主流的应用开发框架的开发模式工作，它不是各个语言应用开发框架的替代者
- 不仅仅只是一款 RPC 框架
  - 内置 RPC 通信协议实现，但它不仅仅是一款 RPC 框架
- 不是 gRPC 协议的替代品
  - 支持基于 gRPC 作为底层通信协议，
  - 在 Dubbo 模式下使用 gRPC 可以带来更好的开发体验，享有统一的编程模型和更低的服务治理接入成本

6.5 Dubbo API实战

Dubbo工作原理



6.5 Dubbo API实战

Dubbo工作原理

- 服务治理控制面
  - Dubbo 治理体系的抽象表达
  - 控制面包含协调服务发现的注册中心、流量管控策略、Dubbo Admin 控制台等，
- Dubbo数据面
  - 集群部署的所有 Dubbo 进程，进程之间通过 RPC 协议实现数据交换
  - Dubbo 定义了微服务应用开发与调用规范并负责完成数据传输的编解码工作。

6.5 Dubbo API实战

Dubbo数据面

- 从两个方面解决微服务实践中的问题
  - 服务开发框架
  - RPC通信协议实现

6.5 Dubbo API实战

Dubbo数据面

- 服务开发框架
  - RPC 服务定义、开发范式
  - RPC 服务发布与调用 API
  - 服务治理策略、流程与适配方式等

6.5 Dubbo API实战

Dubbo数据面

- RPC通信协议实现
  - 从设计上不绑定通信协议
  - 提供了两款内置高性能 Dubbo2、Triple (兼容 gRPC) 协议实现
  - 支持流式通信模型
  - 协议扩展，多协议暴露

6.5 Dubbo API实战

Dubbo服务治理

- 有了服务开发框架，为什么需要服务治理？
  - 在微服务集群环境下，还有一系列问题需要解决
  - 无状态服务节点动态变化、外部化配置、日志跟踪、可观测性、流量管理、高可用性、数据一致性等
  - 这些问题统称为服务治理
  - Dubbo抽象了一套微服务治理模式并发布了对应的官方实现，服务治理可帮助简化微服务开发与运维，让开发者更专注在微服务业务本身

6.5 Dubbo API实战

Dubbo服务治理

- 服务治理抽象
  - 地址发现
  - 负载均衡
  - 流量路由
  - 链路追踪
  - 可观测性
  - 治理生态对API 网关、限流降级、数据一致性、认证鉴权等场景的适配支持。

6.5 Dubbo API实战

Dubbo服务治理

- Dubbo Admin， 提供Dubbo 集群的可视化视图
  - 查询服务、应用或机器状态
  - 创建项目、服务测试、文档管理等
  - 查看集群实时流量、定位异常问题等
  - 流量比例分发、参数路由等流量管控规则下发

6.5 Dubbo API实战

Dubbo核心优势

- 快速易用
  - 多语言SDK
  - 任意通信协议
  - 加速微服务开发，项目脚手架和开发测试

6.5 Dubbo API实战

Dubbo核心优势

- 超高性能
  - 高性能数据传输，内置支持 Dubbo2、Triple 两款高性能通信协议
  - 流式通信
  - 构建可伸缩的微服务集群

6.5 Dubbo API实战

Dubbo核心优势

- 服务治理
  - 流量管控，重试、限流，金丝雀发布等
  - 微服务生态
  - 可视化控制台
  - 安全体系，支持基于 TLS 的 HTTP、HTTP/2、TCP 数据传输通道，并且提供认证、鉴权策略，

6.5 Dubbo API实战

Dubbo核心优势

- 生产环境验证
  - 在阿里内部的应用，取代HSF和Dubbo2
  - 业内更多案例

6.5 Dubbo API实战

Dubbo生态

- 通信协议
  - Dubbo
  - Triple
  - http
  - gRPC
  - Thrift
  - RMI

6.5 Dubbo API实战

Dubbo生态

- 序列化
  - Hessian2
  - Fastjson/Fastjoin2
  - Protobuf
  - Gson
  - MessagePack
  - Avro
  - FST
  - Kryo

6.5 Dubbo API实战

Dubbo生态

- 注册中心
  - Zookeeper
  - Nacos
  - Consul
  - Sofa
  - Polaris
  - Eureka
  - Etcd

6.5 Dubbo API实战

Dubbo生态

- 配置中心
  - Zookeeper
  - Nacos
  - Apollo

6.5 Dubbo API实战

Dubbo生态

- 元数据中心
  - Zookeeper
  - Nacos
  - Redis

6.5 Dubbo API实战

Dubbo生态

- 网关
  - APISIX
  - Shenyu
  - Higress
  - Kong
  - Pixiu
  - Tengine

6.5 Dubbo API实战

Dubbo生态

- 限流降级
  - Sentinel
  - Polaris
  - Hystrix

6.5 Dubbo API实战

Dubbo生态

- 全局事务
  - Seata

6.5 Dubbo API实战

Dubbo生态

- 链路追踪
  - Skywalking
  - Zipkin
  - MicoMeter
  - OpenTelemetry

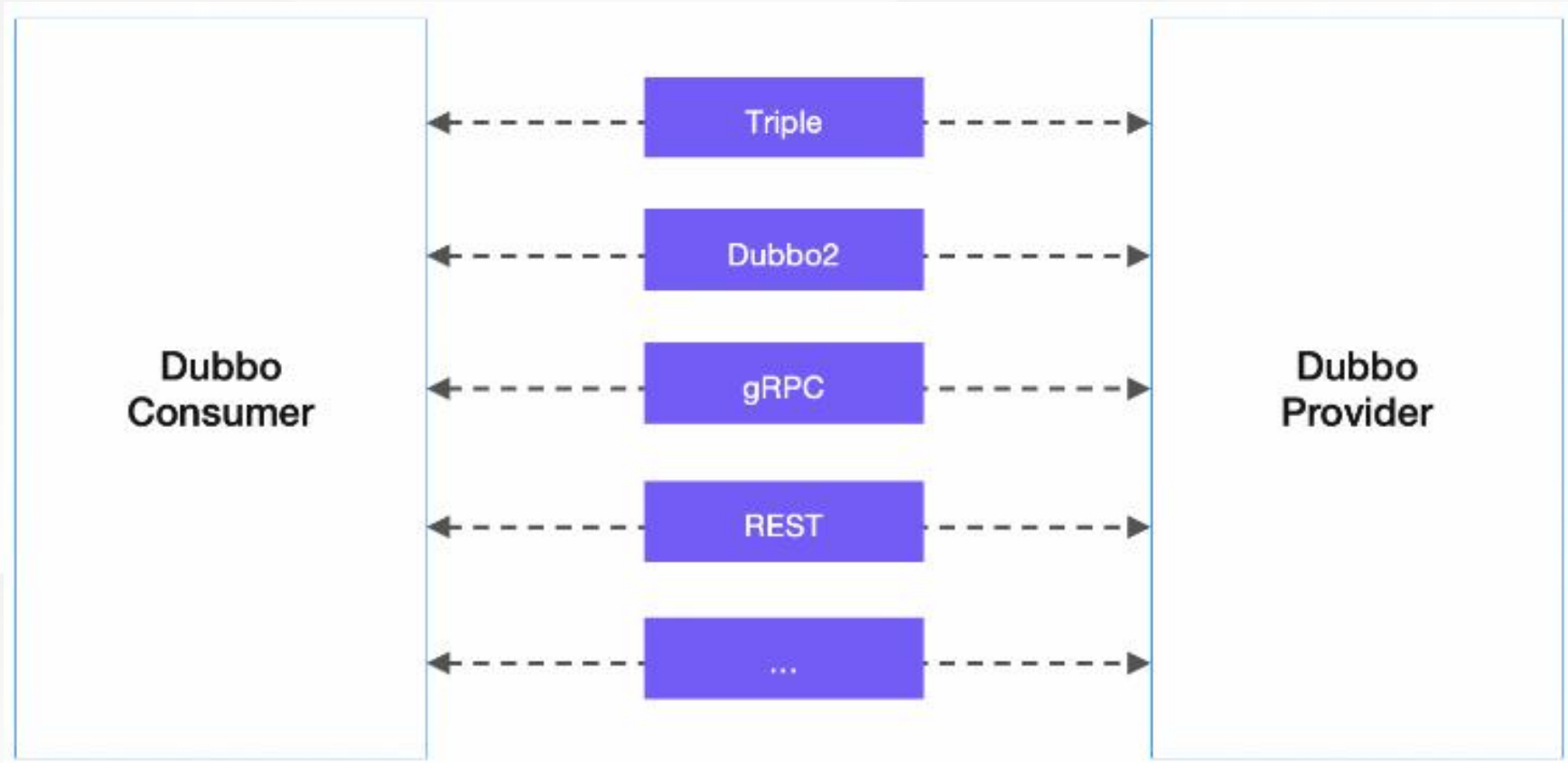
6.5 Dubbo API实战

Dubbo生态

- 监控
  - Prometheus
  - Grafana
  - MicroMeter

6.5 Dubbo API实战

Dubbo 通信协议



6.5 Dubbo API实战

Dubbo 通信协议

- 不绑定任何通信协议，灵活支持多协议，支持在一个应用内发布多个使用不同协议的服务，并且支持用同一个 port 端口对外发布所有协议。
  - 自定义的高性能 RPC 通信协议：基于 HTTP/2 的 Triple 协议 和 基于 TCP 的 Dubbo2 协议
  - Dubbo 框架支持任意第三方通信协议，如官方支持的 gRPC、Thrift、REST、JsonRPC、Hessian2 等，更多协议可以通过自定义扩展实现

6.5 Dubbo API实战

Dubbo 通信协议

- Triple协议
  - Dubbo3 发布的面向云原生时代的通信协议
  - 它基于 HTTP/2 并且完全兼容 gRPC 协议，原生支持 Streaming 通信语义
  - 支持基于 Protobuf 的服务定义与数据传输

6.5 Dubbo API实战

Dubbo 通信协议

- Triple协议
  - 基于 TCP 传输层协议之上构建的一套 RPC 通信协议
  - 提供端异步执行（Server Side Asynchronous Request-Response）
  - 消费端请求流（Request Streaming）
  - 提供端响应流（Response Streaming）
  - 双向流式通信（Bidirectional Streaming）

6.5 Dubbo API实战

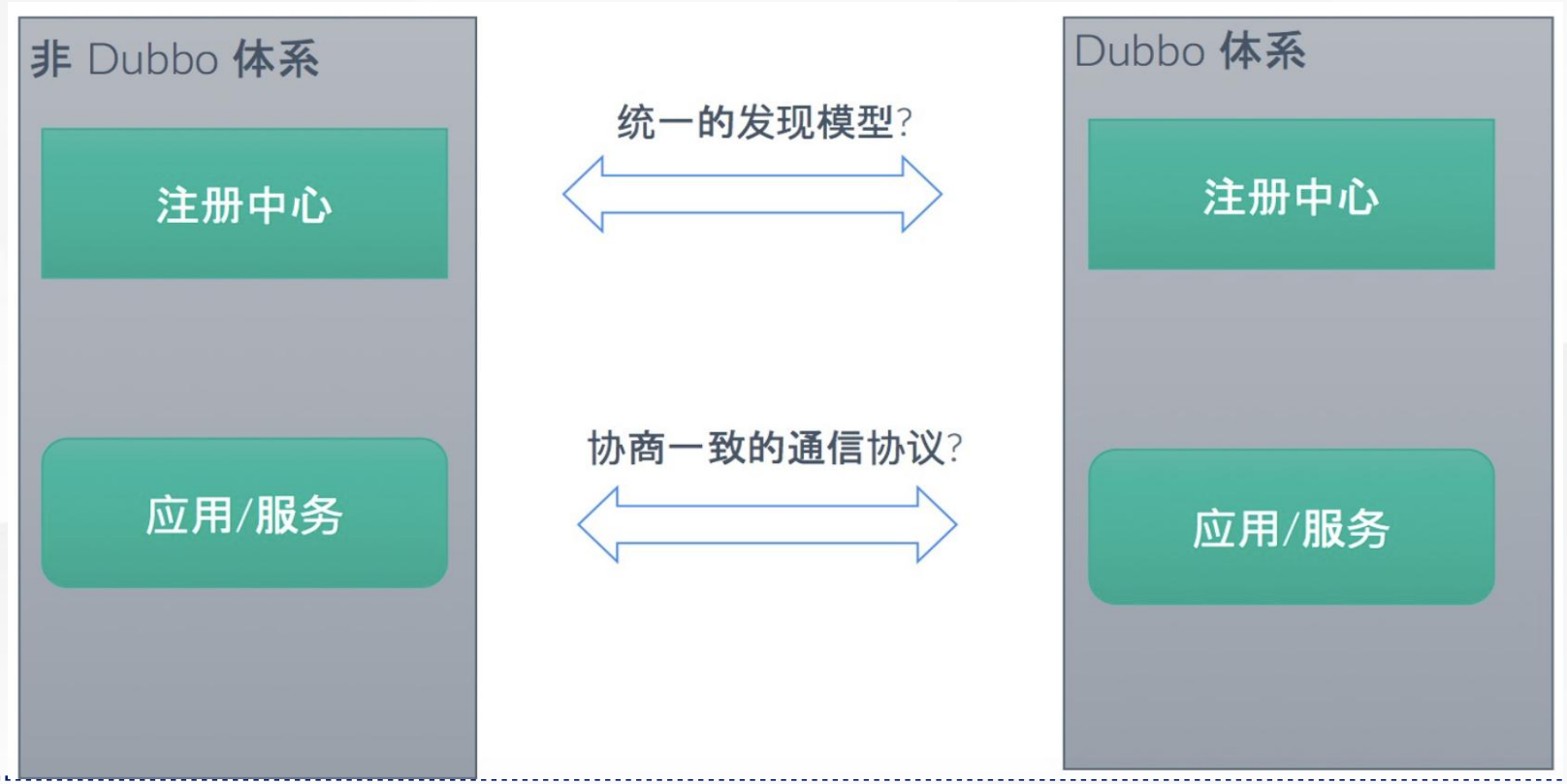
Dubbo 通信协议

- Dubbo2协议
  - 采用单一长连接和 NIO 异步通讯
  - 传输: mina, netty, grizzy
  - 序列化: dubbo, hessian2, java, json
  - 使用范围：传入传出参数数据包较小（建议小于100K），消费者比提供者个数多，单一消费者无法压满提供者，尽量不要用 dubbo 协议传输大文件或超大字符串

6.5 Dubbo API实战

异构微服务互通

◆ 异构微服务体系共存



6.5 Dubbo API实战

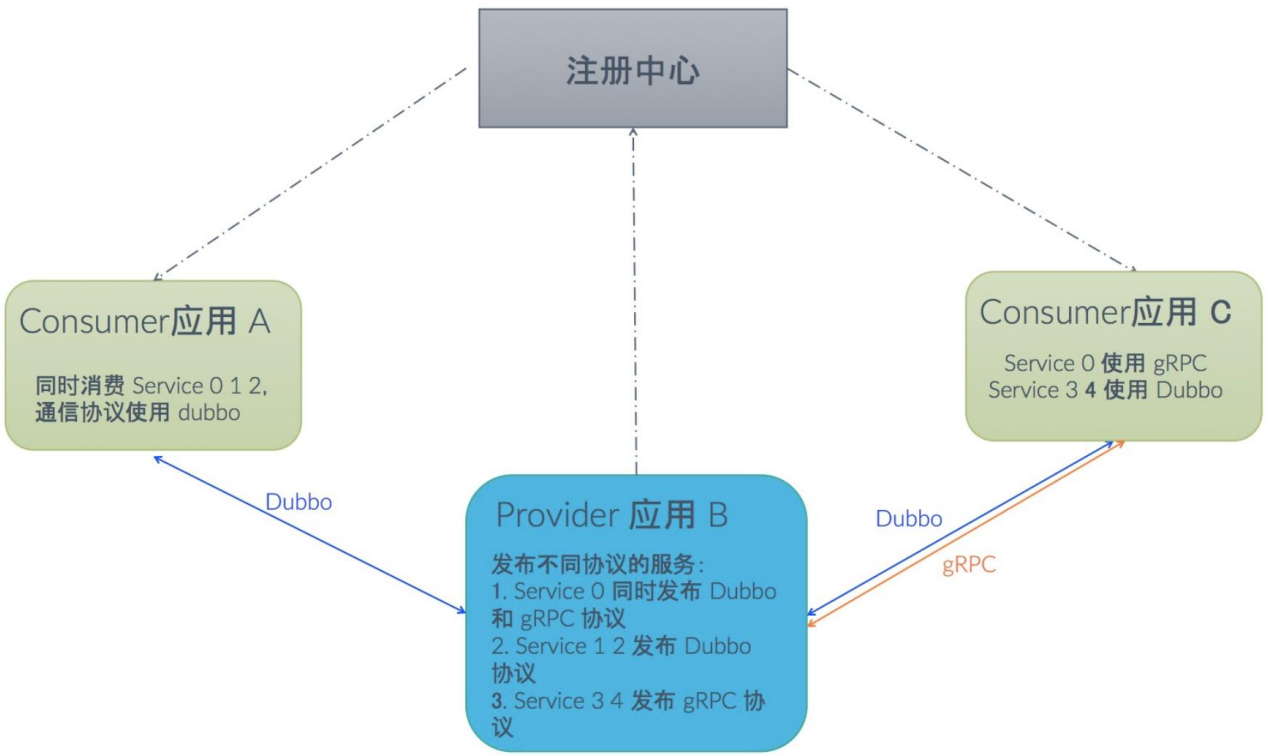
异构微服务互通

- Dubbo体系内部
  - 存在多协议、多注册中心
  - 普通业务采用Dubbo协议
  - 与前端交互需要HTTP协议
  - 流式数据传输业务采用gRPC协议

6.5 Dubbo API实战

异构微服务互通

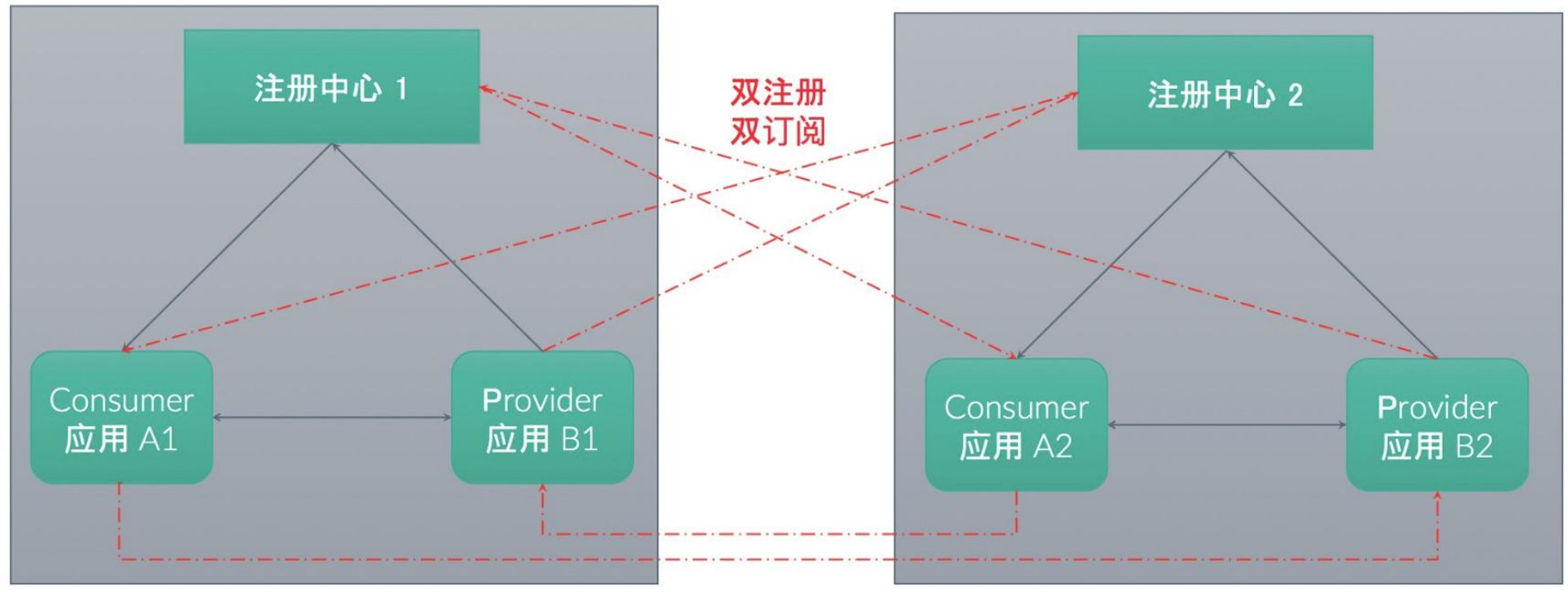
◆ Dubbo的多协议支持



6.5 Dubbo API实战

异构微服务互通

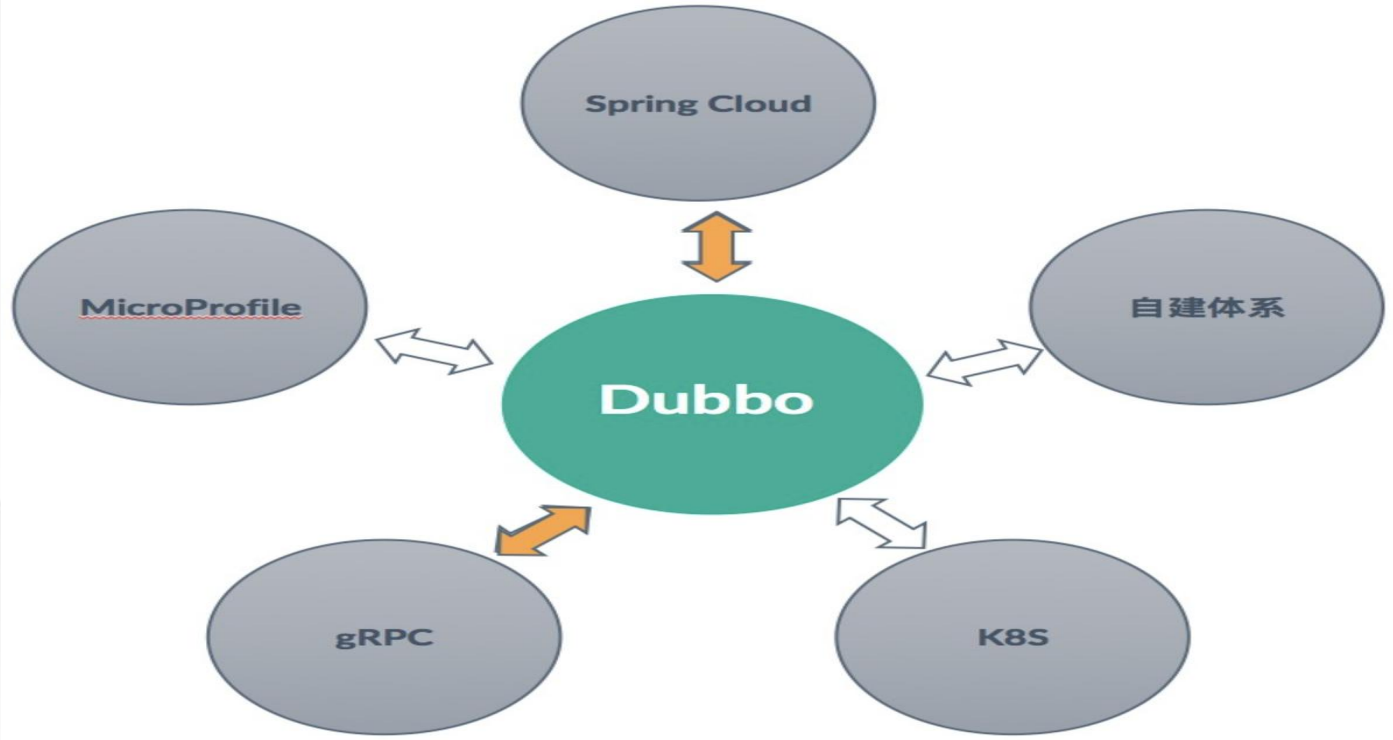
◆ Dubbo的多注册中心支持



6.5 Dubbo API实战

异构微服务互通

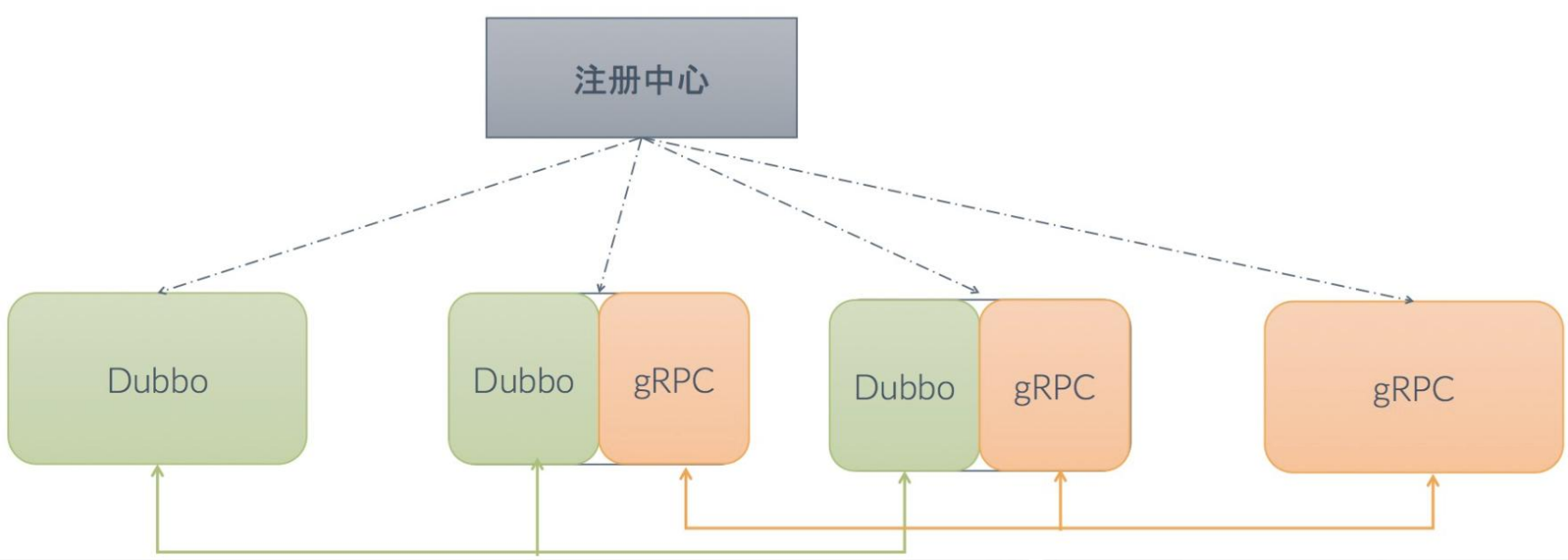
◆借助Dubbo联通异构的微服务体系



6.5 Dubbo API实战

异构微服务互通

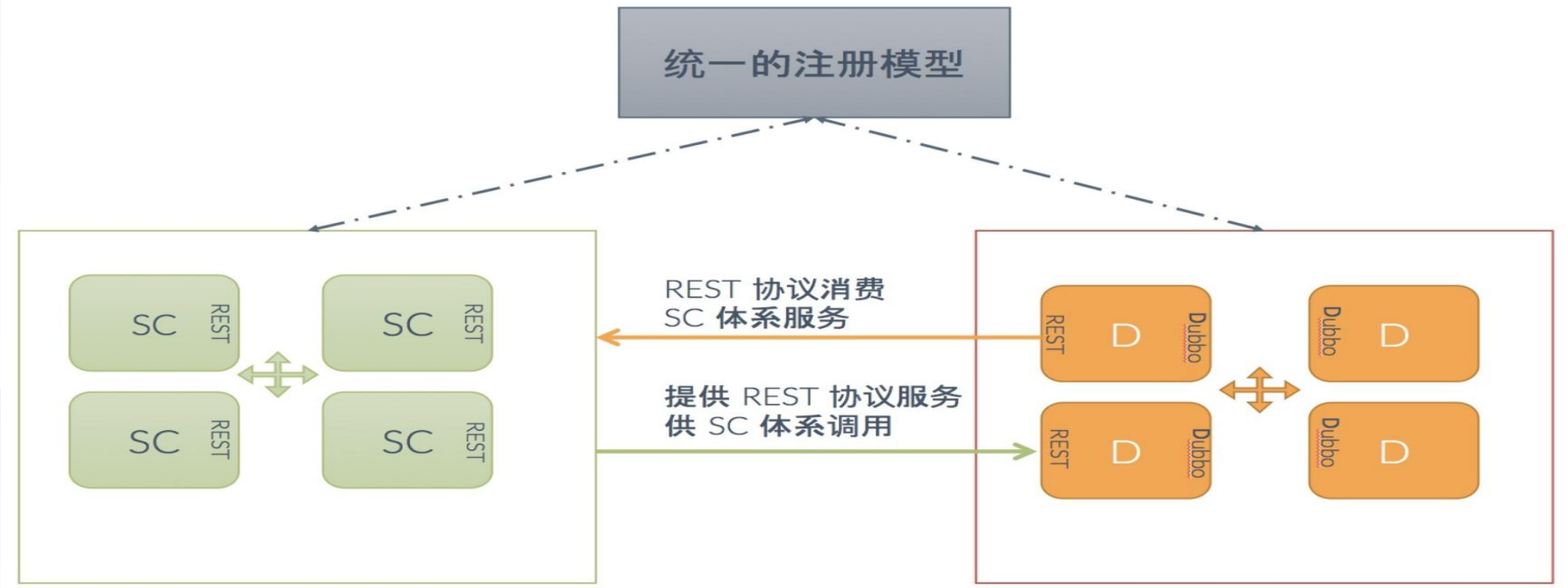
- 借助Dubbo联通异构的微服务体系
  - dubbo体系内的协议迁移



6.5 Dubbo API实战

异构微服务互通

- 借助Dubbo联通异构的微服务体系
  - Spring Cloud 体系迁移到 Dubbo 体系



6.5 Dubbo API实战

Dubbo API案例分析与实现

QA