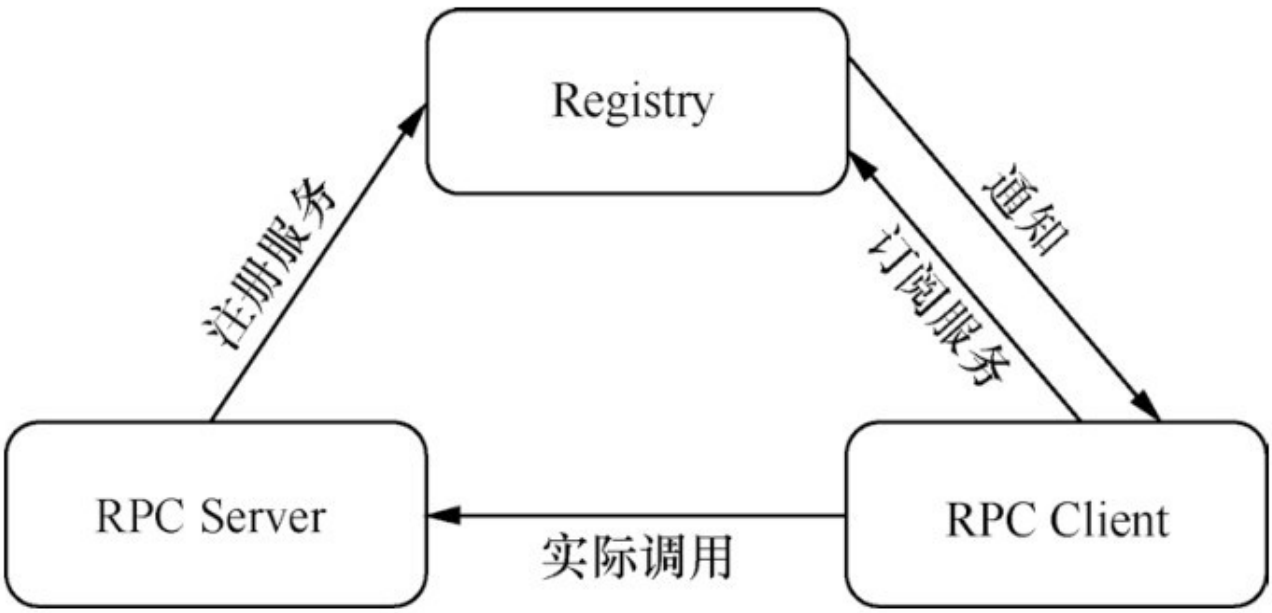
HSF中间件学习笔记

- 参考链接：
  - gitbook： http://mw.alibaba-inc.com/products/hsf/_book/ <http://mw.alibaba-inc.com/products/hsf/_book/>
  - wiki： http://gitlab.alibaba-inc.com/middleware/hsf2-0/wikis/home <http://mw.alibaba-inc.com/products/hsf/_book/>
  - HSF用法详细说明： http://gitlab.alibaba-inc.com/middleware-container/pandora-boot/wikis/spring-boot-hsf <http://gitlab.alibaba-inc.com/middleware-container/pandora-boot/wikis/spring-boot-hsf>

0 RPC 远程调用框架

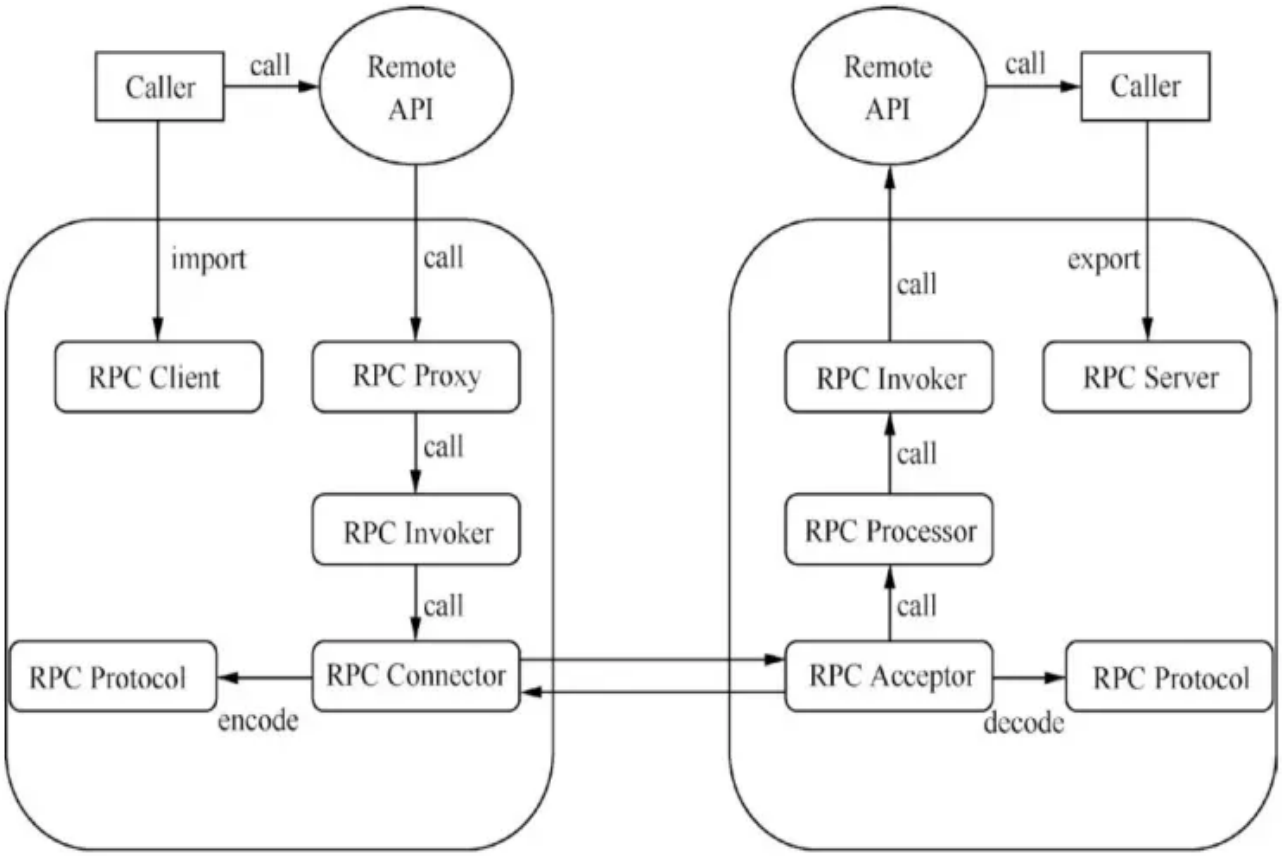
- 参考链接:
  - RPC框架：https://www.zhihu.com/market/pub/119898684/manuscript/1254395671928979456 <https://www.zhihu.com/market/pub/119898684/manuscript/1254395671928979456>
  -

RPC(Remote Procedure Call)框架的本质内容还是，第三方程序的调用，不过他使用的不是本机的第三方程序，而是分布式集群中的其它应用；相当于将任务放到了其它机器上进行执行；RPC框架的关键在于序列化/反序列化、协议编码、网络通信(寻址服务和网络传输);最终实现对于远程调用过程的屏蔽；让操作者感觉是在进行本地调用。



RPC一般的模式都是注册-订阅-服务模式；

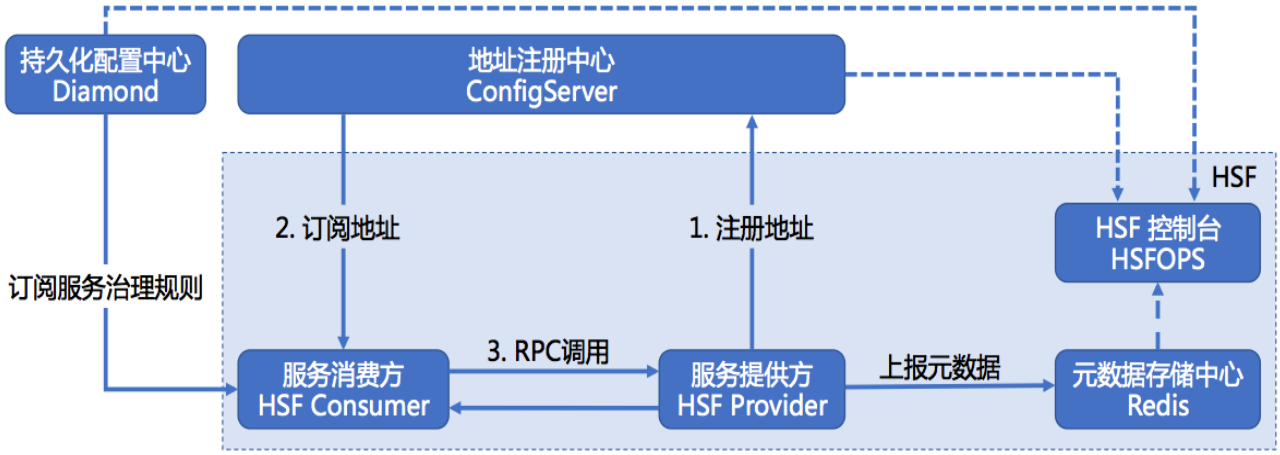
一般RPC的主要底层原理如下:



注意上图的流程中省略了注册和发现中心；其实它也不是必须的
其中典型的RPC框架有:

- grpc：谷歌出品；C++作为主要开发语言；使用protoBuffer 作为序列化的方法；性能高效；使用protobuffer，压缩效率非常高；基于HTTP2，具有流双向流传输等特性；客户端，服务端基于同一份.proto文件来编码不容易出错；支持多种语言。但是文档缺乏；生态简陋；使用C++开发开发成本过高。
- Thrift：一种轻量级的跨语言 RPC 通信方案，支持多达 25 种编程语言。支持多种序列化格式：如 Binary、Compact、JSON、Multiplexed 等。支持多种通信方式：如 Socket、Framed、File、Memory、zlib 等。服务端支持多种处理方式：如 Simple 、Thread Pool、Non-Blocking 等。
- Dubbo：阿里巴巴公司开源的一个Java高性能优秀的服务框架，使得应用可通过高性能的 RPC 实现服务的输出和输入功能，可以和 Spring 框架无缝集成。支持集群容错、自动发现；可以平滑的减少和增加机器。

HSF主要框架如下图:



地址注册证中心

HSF进行服务注册和发现的中心机构，**采用心跳的方式监控各服务运行节点的状况，以便剔除故障的服务提供方。**而注册中心就是服务信息的中介，提供服务发现的能力。在阿里巴巴集团内部，地址注册中心的角色是由 ConfigServer 承担的

持久化配置中心

存储HSF服务的各种治理规则，HSF 客户端在启动的过程中会向持久化配置中心订阅各种服务治理规则，如路由规则、归组规则、权重规则等，从而根据规则对调用过程的选址逻辑进行干预。
在阿里巴巴集团内部，持久化配置中心的角色是由 Diamond 承担的。

元数据存储中心

HSF服务对应的方法列表以及参数结构等信息，主要是为了服务运维的便捷性，HSF客户端在启动时会将元数据上报到元数据存储中心。以便提供给服务运维使用。

HSF控制台

HSF 控制台通过打通地址注册中心 ConfigServer、持久化配置中心 Diamond、元数据存储中心 Redis，为用户提供了一些列服务运维功能，包括服务查询、服务治理规则管理、服务测试、服务 Mock、单机运维等，旨在提高 HSF 服务研发的效率、运维的便捷性。主要就是HSFOPS承担。

整体调用流程

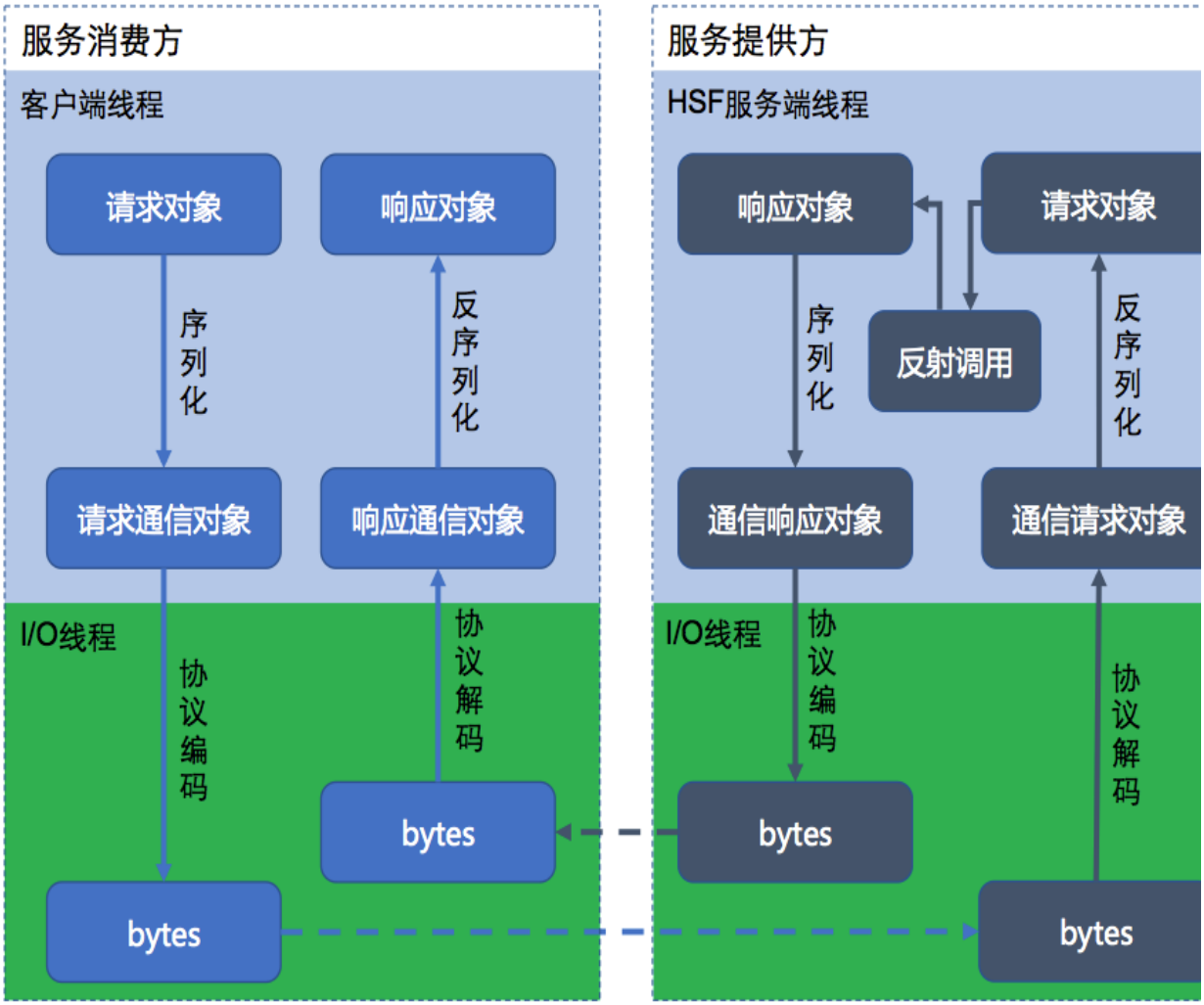
服务方在启动之后会向地址注册中心发布地址，并将元数据上传元数据存储中心，服务消费方根据服务名向地址注册中心订阅服务地址；服务消费方根据订阅服务治理规则选择机器进行rpc远程调用。

HSF特点

- 在网络通信层面，HSF 采用开源的高性能、异步事件驱动的 NIO 框架 Netty 作为网络通信框架。
- 在协议层面，HSF 2.x 做了大量优化，使用了更适用于 RPC 场景的 RPCRemoting 协议替换了 HSF 1.x 中的 TBRemoting 协议，性能有了大幅提高。目前，HSF 在物理机上的压测数据约为 30w QPS。
- 序列化: 使用Hessian2方法进行序列化。

原理解析

其调用过程如下图所示：



服务消费方：

1.HSF客户端线程将请求对象(方法名和参数)序列化并放入通信对象中

2.将通信对象提交给IO线程，编码变成二进制数据

3.通过Netty发送给服务提供方，消费方IO线程等待结果返回，处于等待状态

8.IO线程收到二进制内容，进行解码成通信响应对象

9.唤醒HSF客户端线程，反序列化拿到响应对象

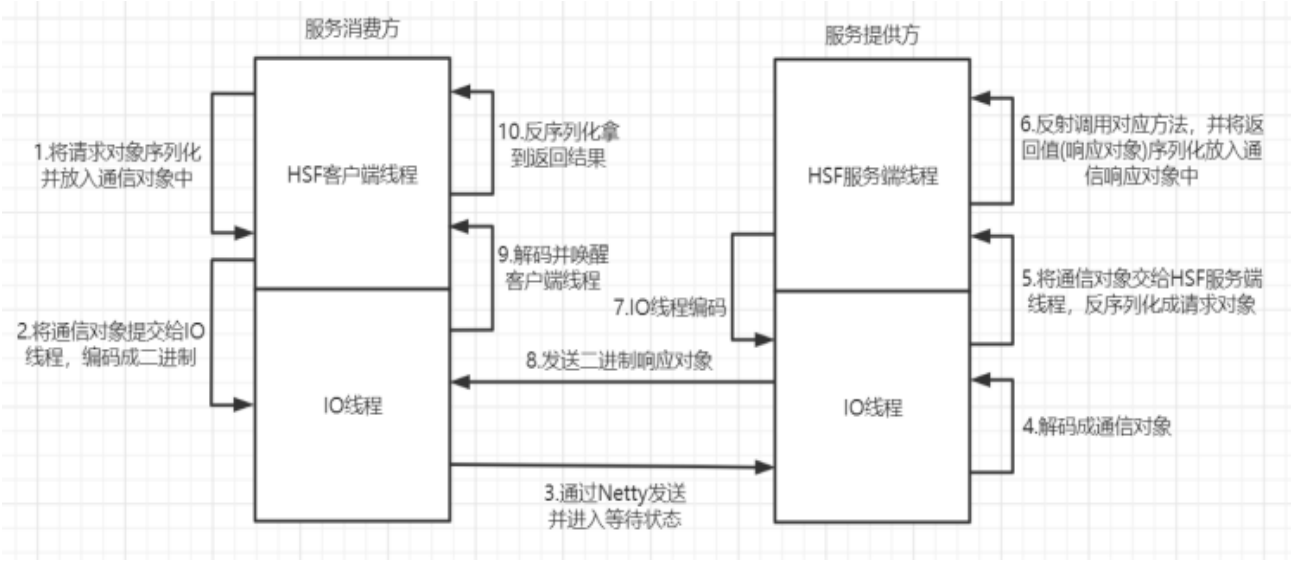
服务提供方：

4.IO线程接收到二进制内容，进行解码成通信对象

5.将通信对象交给HSF服务端线程，反序列化成请求对象

6.反射调用对应方法，并将返回值(响应对象)序列化，并放入通信响应对象中

7.将通信响应对象交给IO线程，编码后发送给服务消费方

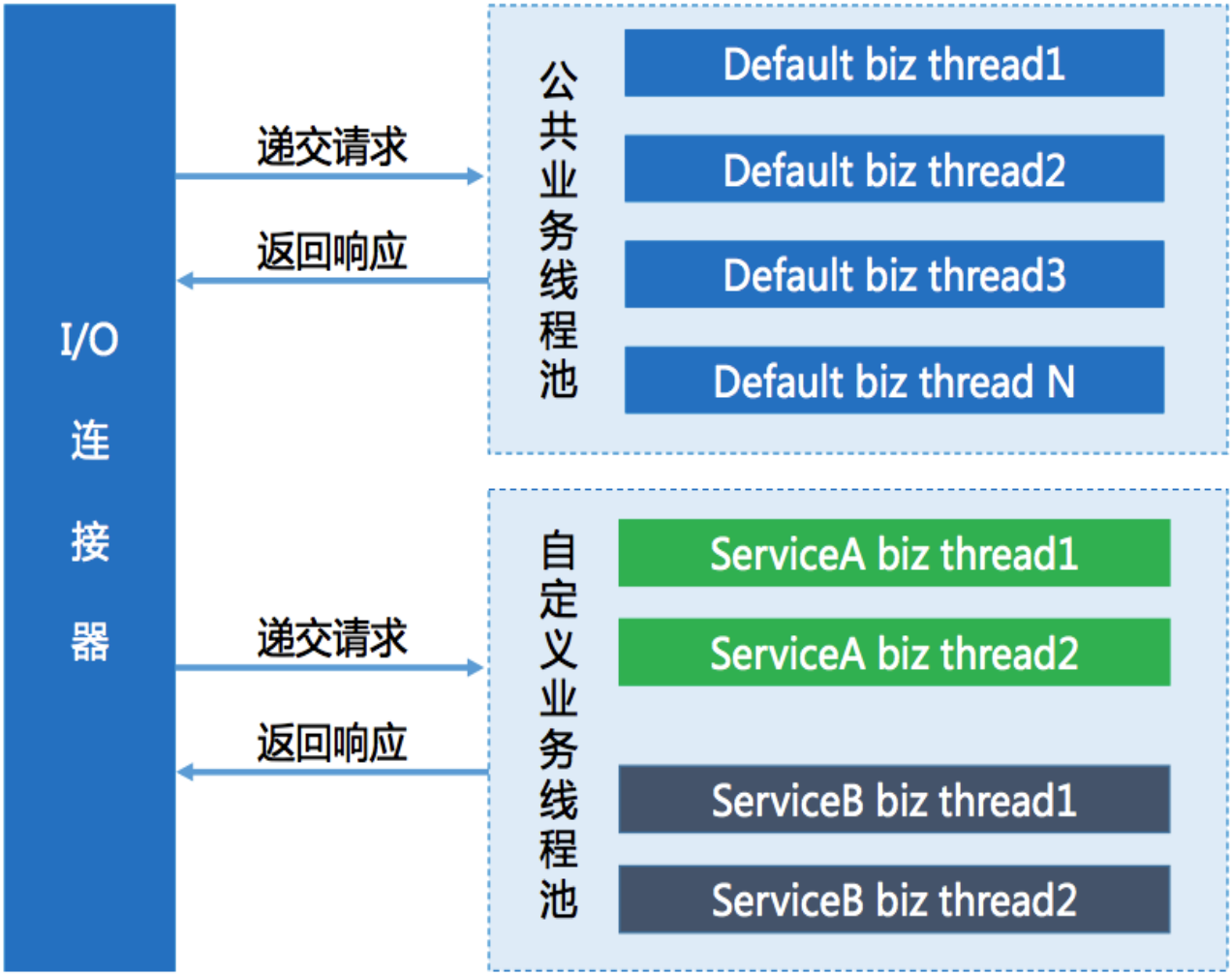


服务调用方式：

1. 同步调用：HSF的IO操作都是异步的，客户端同步调用的本质是做`future.get(timeout)`操作，等待服务端的结果返回，这里的`timeout`就是客户端生效的超时时间(默认3000ms)
2. 异步调用：
   a. Future异步调用：用户可以在上下文中获取跟返回结果关联的`HSFFuture`对象，然后用户可以在任意时刻调用`HSFFuture.getResponse(timeout)`获取服务端的返回结果。
   b. Callback异步调用：客户端配置为callback方式调用时，需要配置一个实现了`HSFResponseCallback`接口的listener，结果返回之后，HSF会调用`HSFResponseCallback`中的方法。

HSF 服务端线程池配置

HSF服务端线程池主要分为IO线程和业务线程，其中IO线程模型就是netty reactor网络模型中使用的。我们主要讨论业务线程池的配置。业务线程池分为默认业务线程池和服务线程池，其中服务线程池是从默认线程池中分割出来的，如下图所示：



1. 服务端接收到了请求对象后，会从线程池服务中获取服务端线程池，用这个线程池完成请求的反射处理
2. 线程池默认的core size是50，max size是720, keepAliveTime 500s。队列使用的是SynchronousQueue，容量1，不会堆积用户请求。 当服务端线程池所有线程都在处理请求时，对于新的请求，会立即拒绝
3. 通过HSF服务端异步处理的目的是不想长时间占据HSF处理线程，而是使用其他线程来完成工作，最后HSF负责将数据写回调用端，这样能够提高吞吐量。
4. 对于一些慢服务、并发高的服务，可以单独为其分配线程池从默认业务线程池中拆分出来，避免占用过多的业务线程，影响其他服务的调用。