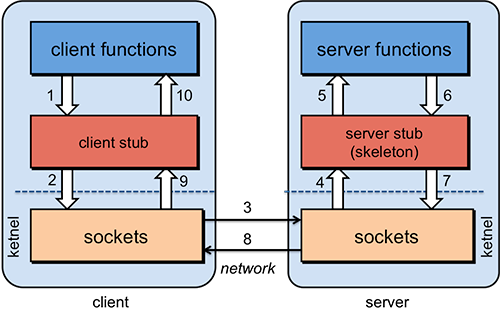
　　RPC简介

RPC（Remote Procedure Call Protocol）——远程过程调用协议，它是一种通过网络从远程计算机程序上请求服务，而不需要了解底层网络技术的协议。RPC协议假定某些传输协议的存在，如TCP或UDP，为通信程序之间携带信息数据。在OSI网络通信模型中，RPC跨越了传输层和应用层。RPC使得开发包括网络分布式多程序在内的应用程序更加容易。

RPC采用客户机/服务器模式。请求程序就是一个客户机，而服务提供程序就是一个服务器。首先，客户机调用进程发送一个有进程参数的调用信息到服务进程，然后等待应答信息。在服务器端，进程保持睡眠状态直到调用信息到达为止。当一个调用信息到达，服务器获得进程参数，计算结果，发送答复信息，然后等待下一个调用信息，最后，客户端调用进程接收答复信息，获得进程结果，然后调用执行继续进行。

下面是一个简单的调用流程图:



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）server stub将消息发送回客户端

9）client stub接收到消息，并进行解码；

10）服务消费方得到最终结果。

其中编码的消息是需要一定规范的，客户端请求消息的数据结构大抵如下:

1. 接口名称

为了让服务端知道是调用哪个接口

1. 方法名

一个接口可能有多个方法，所以需要指定方法名

1. 参数类型及参数值

参数类型有很多，比如有bool、int、long、double、string、map、list，甚至如struct（class），以及相应的参数值

1. 超时时间

表示客户端等待回复时间

1. RequestID 标识唯一请求ID

同理服务端返回的消息结构一般包括以下内容。

1）返回值

2）状态code

3）requestID

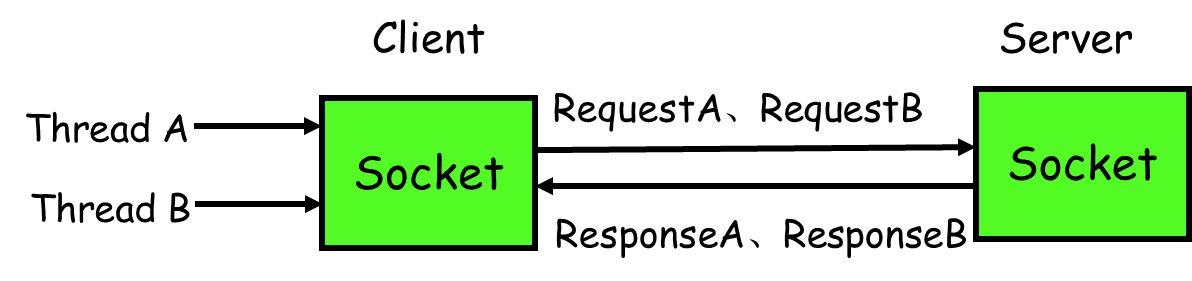
下面是说明一下为什么需要有requestID？

如果远程调用(从发出请求到接收到结果)是异步的，即对于当前线程来说，将请求发送出来后，线程就可以往后执行了，至于服务端的结果，是服务端处理完成后，再以消息的形式发送给客户端的。于是这里出现以下两个问题：

1）怎么让当前线程“暂停”，等结果回来后，再向后执行？次

2）如果有多个线程同时进行远程方法调用，这时建立在client server之间的socket连接上会有很多双方发送的消息传递，前后顺序也可能是随机的，server处理完结果后，将结果消息发送给client，client收到很多消息，怎么知道哪个消息结果是原先哪个线程调用的？

如下图所示，线程A和线程B同时向client socket发送请求requestA和requestB，socket先后将requestB和requestA发送至server，而server可能将responseA先返回，尽管requestA请求到达时间更晚。我们需要一种机制保证responseA丢给ThreadA，responseB丢给ThreadB。



对于这两个问题，requestID可以很好的解决:

1）client线程每次通过socket调用一次远程接口前，生成一个唯一的ID，即requestID（requestID必需保证在一个Socket连接里面是唯一的），一般常常使用AtomicLong从0开始累计数字生成唯一ID；

2）将处理结果的回调对象callback，存放到全局ConcurrentHashMap里面put(requestID, callback)；

3）当线程调用channel.writeAndFlush()发送消息后，紧接着执行callback的get()方法试图获取远程返回的结果。在get()内部，则使用synchronized获取回调对象callback的锁，再先检测是否已经获取到结果，如果没有，然后调用callback的wait()方法，释放callback上的锁，让当前线程处于等待状态。

4）服务端接收到请求并处理后，将response结果（此结果中包含了前面的requestID）发送给客户端，客户端socket连接上专门监听消息的线程收到消息，分析结果，取到requestID，再从前面的ConcurrentHashMap里面get(requestID)，从而找到callback对象，再用synchronized获取callback上的锁，将方法调用结果设置到callback对象里，再调用callback.notifyAll()唤醒前面处于等待状态的线程。

现在我们知道了实现服务，但是如何发布自己的服务给别人用呢?第一反应当然就是直接告诉使用者ip和端口就好啦！但是问题来了，是选择人肉告知还是自动告知，很明显后者更好。zeekeeper实现了自动告知，即机器的增添、剔除对调用方透明，调用者不再需要写死服务提供方地址。

zookeeper就是个分布式文件系统，每当一个服务提供者部署后都要将自己的服务注册到zookeeper的某一路径上，zookeeper提供了“心跳检测”功能，它会定时向各个服务提供者发送一个请求（实际上建立的是一个 socket 长连接），如果长期没有响应，服务中心就认为该服务提供者已经“挂了”，并将其剔除。

服务消费者会去监听相应路径（/HelloWorldService/1.0.0），一旦路径上的数据有任务变化（增加或减少），zookeeper都会通知服务消费方服务提供者地址列表已经发生改变，从而进行更新。更为重要的是zookeeper 与生俱来的容错容灾能力（比如leader选举），可以确保服务注册表的高可用性。

资源链接:

1. RPC

Rpcx: <https://github.com/smallnest/rpcx>

Rpcx文档: <https://www.gitbook.com/book/smallnest/go-rpc/details>

Go-micro: <https://github.com/micro/go-micro>

Go-micro文档: <https://micro.mu/docs/>

Dubbo: <https://github.com/alibaba/dubbo>

Grpc: <https://godoc.org/google.golang.org/grpc>

Protobuf go插件: <http://github.com/golang/protobuf/protoc-gen-go>

入门教程: <https://grpc.io/docs/quickstart/go.html>

二．管理框架

Zookeeper: <http://www.apache.org/dyn/closer.cgi/zookeeper/>

Zooeeper文档翻译版: <http://blog.csdn.net/kobejayandy/article/details/11836051>

Etcd: <https://github.com/coreos/etcd>

Etcd文档: <https://coreos.com/etcd/docs/latest/>