RPC架构里包含如下4个组件:

1、 客户端(Client)：服务调用方

2、 客户端存根(Client Stub)：存放服务端地址信息，将客户端的请求参数打包成网络消息，再通过网络发送给服务方

3、 服务端存根(Server Stub)：接受客户端发送过来的消息并解包，再调用本地服务

4、服务端(Server)：真正的服务提供者。

具体实现步骤：

1、 服务调用方（client）(客户端)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；在Java里就是序列化的过程

3、 client stub找到服务地址，并将消息通过网络发送到服务端；

4、 server stub收到消息后进行解码,在Java里就是反序列化的过程；

5、 server stub根据解码结果调用本地的服务；

6、 本地服务执行处理逻辑；

7、 本地服务将结果返回给server stub；

8、 server stub将返回结果打包成消息，Java里的序列化；

9、 server stub将打包后的消息通过网络并发送至消费方

10、 client stub接收到消息，并进行解码, Java里的反序列化；

11、 服务调用方（client）得到最终结果。

使用RPC远程服务调用方式与传统http接口直接调用方式的差别在于：

1. 从使用方面看，Http接口只关注服务提供方（服务端），对于客户端怎么调用，调用方式怎样并不关心，通常情况下，客户端使用Http方式进行调用时，只要将内容进行传输即可，这样客户端在使用时，需要更关注网络方面的传输，比较不适用与业务方面的开发；而RPC服务则需要客户端接口与服务端保持一致，服务端提供一个方法，客户端通过接口直接发起调用，业务开发人员仅需要关注业务方法的调用即可，不再关注网络传输的细节，在开发上更为高效。

2. 从性能角度看，使用Http时，Http本身提供了丰富的状态功能与扩展功能，但也正由于Http提供的功能过多，导致在网络传输时，需要携带的信息更多，从性能角度上讲，较为低效。而RPC服务网络传输上仅传输与业务内容相关的数据，传输数据更小，性能更高。

3. 从运维角度看，使用Http接口时，常常使用一个前端代理，来进行Http转发代理请求的操作，需要进行扩容时，则需要去修改代理服务器的配置，较为繁琐，也容易出错。而使用RPC方式的微服务，则只要增加一个服务节点即可，注册中心可自动感知到节点的变化，通知调用客户端进行负载的动态控制，更为智能，省去运维的操作。

从使用方面看，Http接口只关注服务提供方（服务端），对于客户端怎么调用，调用方式怎样并不关心，通常情况下，客户端使用Http方式进行调用时，只要将内容进行传输即可，这样客户端在使用时，需要更关注网络方面的传输，比较不适用与业务方面的开发；（restful是服务端把方法写好，客户端通过http方式调用，直接定位到方法上面去。）

而RPC服务则需要客户端接口与服务端保持一致，服务端提供一个方法，客户端通过接口直接发起调用，业务开发人员仅需要关注业务方法的调用即可，不再关注网络传输的细节，在开发上更为高效。（PRC是服务端提供好方法给客户端调用。定位到类，然后通过类去调用方法。）



比如这种，自己要了一个计算服务，拿到计算服务类后，自己调用服务类里的加法去获得结果

如果是restful，就根据Calculate方法对应的url去传参（people），从而获得结果。

RPC：所谓的远程过程调用 (面向方法)

SOA：所谓的面向服务的架构(面向消息)

REST：所谓的 Representational state transfer (面向资源)

RPC 即远程过程调用, 很简单的概念, 像调用本地服务(方法)一样调用服务器的服务(方法).

通常的实现有 XML-RPC , JSON-RPC , 通信方式基本相同, 所不同的只是传输数据的格式.

REST 的三个要素是 唯一的资源标识, 简单的方法 (此处的方法是个抽象的概念),一定的表达方式.

重要的特性：无状态