分布式RPC框架如何进行服务寻址和分发

December 13, 2014

所谓分布式，就是一个服务可能分布在多个Server上。另外，一个Server往往有多个Service，而每个Service有多个方法（服务）。为了对业务透明，服务框架一般会屏蔽远程分布式调用的细节，让用户觉得这就是一个本地服务，也就是说客户端可能就是这样一个调用：

```
String result = HelloService.sayHello();
```

至于这个HelloService的sayHello()服务是部署在哪里，它并不关心。但是对于框架，它却需要一定的信息来对这个服务进行寻址和分发。

首先是服务寻址，需要知道XXXService是部署在哪里。也就是说需要得到一个 [<host, port>] 的服务地址列表。然后客户端负载均衡选择一个进行请求。为了动态增加服务不影响客户端，这个职责一般归属在一个叫做配置中心的系统。服务提供者像配置中心注册它提供的服务，服务消费者查询某个服务的提供者。也就是说注册中心有两个核心的方法:

```
boolean register(key, serviceAddress);
List<String> getAvailableServiceAddress(key);
```

那么关键在于这个key是什么呢？

一般有两种实现方案：

1. 采用接口的全名。比如me.arganzheng.study.rpc.HelloService。
2. 采用分配的命令字。比如一个int32，这个相对于上面的做法稍微人肉一些。

TIPS

1. 考虑到接口的升级问题，一般会加上版本号。
2. 考虑到跨机房和容灾问题，可能还需要考虑按照set部署，那么还需要加上一个set号。
3. 由于这个key还需要加上方法名传递到服务端让服务端进行本地的服务分发，如果采用接口全名的话，key会比较大，而且是不定长的。可以考虑将key做一个md5。

经过注册中心寻址到服务提供者之后，RPC框架发送一个请求到服务提供者，然而前面说过了，一个Server可能部署很多个Service，而且每个Service可能有多个服务。所以这里Server还需要根据一定的信息做一个本地的服务分发：

```
Object service = getServiceByNameAndVersion(req.serviceName, req.serviceVersion);
Object result = invokeMethod(service, req.methodName, req.args);
```

根据前面的key，我们能够找到相应的service。然后根据method，我们能够知道要调用该service的哪个方法。

注意到这里其实有个小小的矛盾：服务其实就是方法，所以理论上来说整个服务的寻址和分发应该是方法粒度。但是由于ServerAddress只能是Socket粒度，所以注定了配置中心只能返回Server粒度，然后Server接到请求之后再做更细粒度的服务分发。

也就是说，其实我们可以把key定义为方法级别的，比如：

```
me.arganzheng.study.rpc.FooService.v1 => 220.181.57.215:8090
me.arganzheng.study.rpc.FooService.v2 => 220.181.57.216:8090
```

变成

```
me.arganzheng.study.rpc.FooService.foo.v1 => 220.181.57.215:8090
me.arganzheng.study.rpc.FooService.bar.v1 => 220.181.57.215:8090
me.arganzheng.study.rpc.FooService.foo.v2 => 220.181.57.216:8090
```

可以看到，改成方法级别之后，key变多了，但是确实控制粒度更细腻了。升级一个方法，不需要整个service的版本都变化。 不过对于Java这种方法依附于类的语言，升级一个方法，其实整个Service都需要发布。所以实际上没有什么关系。

Related Posts

- 24 Jul 2015 » Tomcat调优
- 22 Jul 2015 » 记一次MySQL主从同步错误处理
- 03 Jul 2015 » Metric监控系统