Config Server和SLA在RPC中的作用

February 2, 2014

背景

在分布式系统中，经常会遇到这样的业务场景：

1. 查找远程服务的地址
2. 查找资源地址（如mc，redis等）
3. 访问远程服务时，获取超时时间、Load balance策略、灰度策略、HA策略等配置

当系统规模较小时，地址信息和配置信息可以保存在本地的文件系统中，以达到简单高效的目的。但是随着系统的壮大和服务器数量的增加，配置和地址信息的变更可能需要重启大量的服务器，这将会是十分“重量级”的操作。而这就是Config Service所要解决的问题。

Config Service

Config Service是集配置服务和命名服务于一身的基础服务。它将地址信息和配置信息集中管理在云端，并且提供实时的变更通知。最常见的使用方是RPC服务。原则上来说，系统中发生的每一次远程调用，都要经过配置中心(或者本地agent)来寻址；系统中每一处的配置，都要从配置中心(或者本地agent)获取。因此，如果把分布式系统比喻为一张网的话，那么配置中心就处于这张网的中心。

实现

对外接口来看，配置中心其实就是一个简单key-value配置信息存储和查询的服务。服务提供方把配置信息存储到config service中，服务调用方根据约定好的key进行查询，得到value进行相应的处理。那么像memcached或者redis，甚至MySQL，这样的key-value store就可以完成了（新浪微博使用redis作为底层存储）。但是，正如前面所说的，原则上所有的RPC和配置信息都需要走Config Service，这将会对系统处理能力和网络带宽带来无法想象的灾难。而且，config service容易成为单点，一旦挂了，后果无法想象。所以，基本上，所有的Config Service都会引入本地缓存(config agent) 来解决这两个问题（性能和单点问题）。但是本地缓存有个最大的问题，就是一致性问题：当服务端的配置信息变更之后，需要把本地agent的信息同步更新。为了解决这个问题，又需要引入一个通知机制: pub-sub 配置信息变更通知，其实就是GoF中的观察者模式：

1. config agent在启动时候连接到config service，进行注册。
2. config service推送配置信息给config agent。

这需要agent和service之间保存长链接。而且这个长链接其实大部分情况下是空闲的，所以一般还会引入NIO进行线程复用。如果不想引入长链接，那么就可以采用定期轮询方式，不过这种方式，一般是全量更新，无法实现只更新变更部分。新浪微博的Vintage就是这么做的。

另外，config agent也可以根据负载均衡情况反馈给config service，比如某个节点总是调用不

通，其实就是挂掉了，如果config service有提供写接口给agent，那么agent可以更新service的信息，通知到其他agent。

综上，一般来说Config Service可以这样子实现：

1. key-value 配置信息存储
2. config agent 本地缓存配置信息，并且进行Load Balance。
3. pub-sub 配置信息变更推送

关于服务节点的摘除与恢复

服务节点一般在启动时候自动注册，当然也可以人工进行，这个成本不高。关键在于后续的节点状态变更。如果config service如何检测到一个服务节点挂掉，应该把它摘除呢？一般来说有如下两种方式：

1. 服务节点通知config service
    - 服务节点定期向config service发送心跳信息，告诉config service它还活着。但是由于config service和服务节点间的连接是不可靠的，服务节点可能会被误认为是死节点（典型的Two Generals Problem）。
    - 服务节点与config service之间保存长链接，如果断开了，表示服务节点不可用。不过同样会有网络间接不可靠问题存在。
2. config agent(代表服务调用方)通知config service
    - 服务调用方通过config agent查询到服务提供者列表，根据负载均衡策略得到选定的服务提供者，根据请求的响应时间和成功率可以判断服务提供者是否是正常工作。这些信息可以通过config service的写接口反馈给config service。

不过正如前面所说，不能百分百确定服务节点不可用。是不是要把检测到的“死”节点摘除是个问题？另外，摘除之后的检测恢复也是要考虑的。

新浪微博的做法是将这个问题的选择权交给了查询方。Vintage会把死节点标记为unreachable，且和working的节点一同返回给查询方，而查询方可以选择如何处理unreachable的节点。 ECC的L5是每个config agent自己根据调用情况进行摘除和检测恢复，但是不反馈给config service。config service的职责弱化为key-value store，基本上用于配置信息的初始化而已。是一个典型的去中心化实现。

请求路由

负载均衡与灰度发布

Config Server根据请求的命令号识别出可以处理该请求的BizServer。包括灰度策略。有以下路由策略：

- 按一致性哈希路由：将不同用户的请求比较均匀地摊分到各个业务服务实例去。
- 按号段路由：将沙盒号段用户请求路由到指定的沙盒服务器上。
- 按版本号路由：也是一种灰度策略，特定版本区间的客户端的请求路由到指定机器。

请求重试与故障屏蔽

Config Server路由请求给某台BizServer，但无法被处理时，可以根据配置信息自动将请求转给另一BizServer。另外，同一BizServer错误多次后Config Agent会有个策略对其进行屏蔽和解除屏蔽。

Config Service in RPC

话说现在大多数企业还是习惯把配置信息存放在本地配置文件中，因为配置信息基本上是比较静态的。但是地址信息就相对动态多了，因为服务的线性拓展和负载均衡是很常见的，所以一在RPC框架中，基本上会引入Config Service提供服务的注册和查询服务。

不过笔者在实践中发现，在RPC中，Config Service仅仅提供地址服务是不够的，因为各个服务的SLA（Service Level Agreement）各不相同，而这个信息对于负载均衡非常重要，这些信息包括 超时时间、失败容忍度、频率限制、自动服务降级或者手动关闭，QoS，同步异步、读写接口等等。这样，客户端可以根据这些信息进行更好的Load Balance和Invoke。其实现在API元数据已经逐渐承当了这部分角色，配合Config Service的地址服务，能够更好进行RPC调用。

参考文章

1. Config Service - 微博引擎的点火器
2. 微博平台服务保障与SLA

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