通用运维流程架构

By @CodeBox-腾讯

设计理念

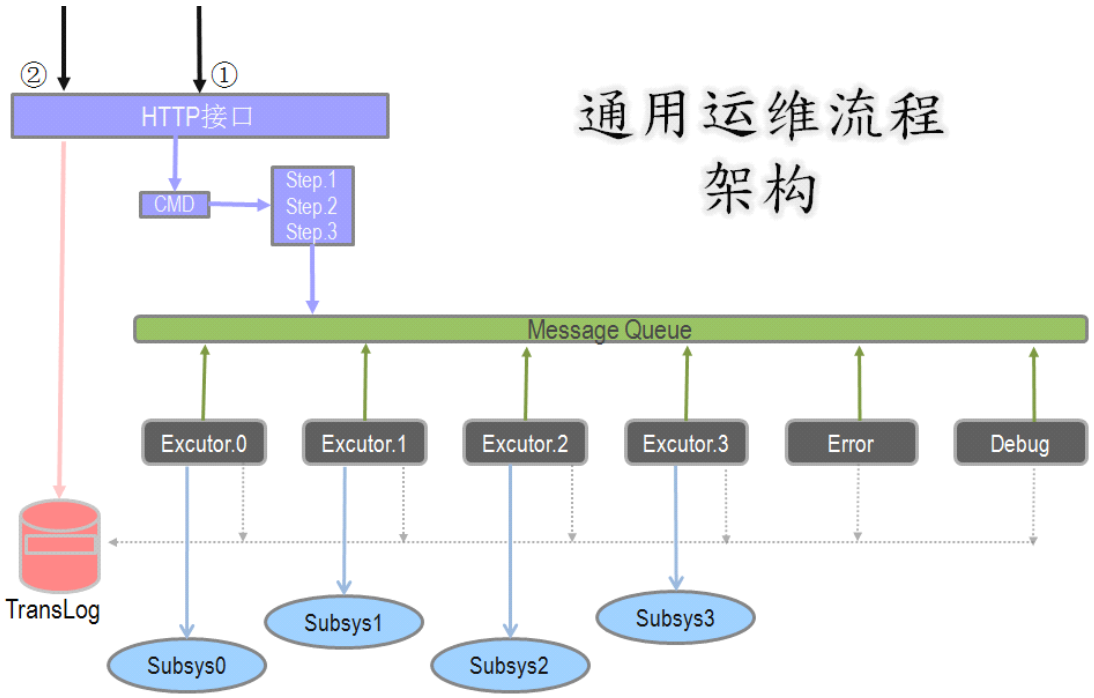
1. 简单。
2. 可扩展。
3. 开发量少。
4. 无状态。
5. 容易问题定位。

设计目标

1. 单一性，每个运维子系统提供单一最纯粹的服务，比机器资源子系统，域名子系统。
2. 原子性，每个子系统的只提供最原始的原子接口，即只提供机制，不提供策略。
3. 幂等性，一个接口被多次调用，不会产生副作用。
4. 松耦合，每个运维子系统不知道其它子系统的存在，最大程度上解耦。
5. 无状态，架构中的每个点都是无状态的，可以任意水平扩展。
6. 统一协议，每个运维子系统提供的都是 http+json 的接口。

架构解析

架构图如下：



Web 层

紫色部分是整个运维系统提供的 http 功能性接口，供 web 页面或其它系统调用。

设计思路是这样的 :接收到大的功能请求后 ,将其拆成内部子系统(蓝色部分)的原子接口，并根据流程，排列成有序的步骤(图中将 cmd 拆解成 step0,1,2)。将其作为一个消息投递的绿色的消息总线中，然后 http cgi 就给调用方返回了(返回一个 taskid,供调用方实轮询 task 运行结果)。

为方便前端调用，cgi 可以将以上两个步骤合起了，形成一个同步接口。

消息队列

绿色部分是消息队列，是驱动整个系统运转的消息总线。

执行器

黑色部分是原子接口执行器 excutor ,每个 excutor 对应一个运维子系统(蓝色部分)。Excutor 监听消息总线中属于自己的消息，消息由 1 中的 cgi 投递或者其实它上游的 excutor 投递。Excutor 收到属于自己的消息后，进行个性化的逻辑处理，处理完成后将处理结果写到 translog 中，供 cgi 查询 ;最后 excutor 将消息重新投放到消息队列中，接收者为下一步的 excutor，直到完成。

有两个比较特殊的 excutor，Error 和 Debug。所有的常规 excutor 执行发生错误，会将消息的接收者设置为 Error，Error 会把消息置成回滚状态 ,重新投递到消息队列中去。这个时候的执行顺序就是反着的了，step2,step1,step0，依次执行回滚操作。这样 Error 的消息可以作为观察系统异常的窗口，不用到各个子系统上到处查日志什么的，便于定位问题。

消息队列中的所有消息都有一份走到 Debug 中，用于开发调试，同样是一个观察整个大系统运行的窗口

TransLog

红色的 translog 是 excutor 的执行结果的保存，cgi 根据 taskid 来轮询任务的完成情况。

运维子系统

蓝色的是运维子系统，是提供具体特定服务的运维系统。其内部的总是设计实际上也是正在介绍的这套框架，也就是说像是递归一样。到最后一层的子系统，基本上就是操作系统库，或跟 RS 机器的各种 agent 打交道了。

采用的组件

1. web 层 Qzhttp

2.消息队列 RabbitMQ 的 worker 模式

3.Excutor 用 python 实现的框架

4.TransLog 用自研的持久化内存存储，接口兼容 memcache

5.有些事务必须加锁的，使用 zookeeper。

这样，每个结构都可以做集群来提高性能和稳定性。

后续给大家介绍特定子系统相关内裤。