

www.qconferences.com
www.qconbeijing.com



伦敦 | 北京 | 东京 | 纽约 | 圣保罗 | 上海 | 旧金山
London · Beijing · Tokyo · New York · Sao Paulo · Shanghai · San Francisco

QCon全球软件开发大会

International Software Development Conference



百度的运维客户端技术

——历史、设计原则与常见误区

钟溢原

yiyanzhong+qcon2013@gmail.com



- ✓ 百度拥有国内首屈一指的服务器数量
- ✓ 全自动运维平台NOAH
 - 设计容量管理百万级服务器
 - 有能力同时操作10k数量级服务器
 - 常见的服务器级故障自动修复能力
 - 日常操作不需要人类登录
 - 提供部署、监控、日志、备份、故障排查、安全审计等





客户端的背景



- ✓ 我们面临的问题从来就没有有什么大的变化，唯一不同的只是机器规模越来越大、人心越来越复杂。



- ✓ 一切都源于那个亘古不变的真理：扔个文件上去，然后跑个命令。

```
# scp destroy.sh www.baidu.com:/root
```

OK

```
# ssh www.baidu.com 'sh /root/destroy.sh'
```

Baidu destroyed successfully.

```
# :-)
```



✓ 后来百度有了10台机器。

```
# for i in `seq -w 1 10`; do scp destroy.sh  
www-$i.baidu.com:/root; ssh www-$i.baidu.com  
'sh /root/destroy.sh'; done
```

Baidu-01 destroyed successfully.

Baidu-02 destroyed successfully.

...

Baidu-10 destroyed successfully.

Baidu destroyed successfully.

```
# :-)
```



✓ 再后来百度有了1000台机器。

```
# for i in `seq -w 1 1000`; do scp destroy.sh  
www-$i.baidu.com:/root; ssh www-$i.baidu.com  
'sh /root/destroy.sh'; done
```

Baidu-0001 destroyed successfully.

Baidu-0002 destroyed successfully.

Wait, are you kidding me?!

Baidu is still alive.

```
# :-0
```



- ✓ 于是百度就有了第一个客户端。
 - 批量自毁的客户端
 - 并发控制、失败重做、总体进度展示、历史记录保存.....

（事实上，百度的第一个客户端是用于监控目的，而且并不接受远程控制命令。）



- ✓ 然后百度就有了第100个客户端。
 - 批量自毁的客户端
 - 监控批量自毁的客户端的客户端
 - 监控监控批量自毁的客户端的客户端的客户端
 - 利用CPU热量来煎鸡蛋的客户端
 - 占用50%内存但是什么都不做的客户端
 - And a lot more...



客户端的背景



百度客户端发展史



- ✓ 百度的公司级客户端17款，兼容3种机型、8种发行版。
 - 系统完整性检查和包管理
 - 机器健康监控
 - 业务本地监控
 - 远程代码执行
 - 定时任务
 - 名字服务
 - And a lot more...

- ✓ 此外，还有公司以下级客户端若干。



- ✓ 第一阶段：人工干预的时代
- ✓ 第二阶段：各自为政的自动化时代
- ✓ 第三阶段：遵循规范的自动化时代
- ✓ 第四阶段：差异化的时代



- ✓ 在这个时代，客户端的设计和维持往往是针对特定产品线的需求而进行定制，客户端没有通用性和扩展性的概念。由于只在自己产品线内使用，往往也没有设计授权机制。机器数量的稀少意味着故障也少，维护成本低，因此自动化程度也低。
 - 客户端是通过ssh批量安装的
 - 升级是一个很大的问题
 - 没有安全机制
 - 每当客户端崩溃的时候，求root密码

- ✓ 代表作：NOAH Client 1.0
- ✓ NOAH历史最悠久的客户端，承担机器健康监控，每隔10-60秒将单机CPU、内存、硬盘等信息发往中心服务器。
- ✓ 部署方法：人肉ssh。



- ✓ 机器数量上升、乐于使用的产品线增加，使得原来认为不是问题的地方突然出现了严重的问题。为了照顾不同产品线的需求差异，客户端要求具有较强的通用性。由于出现多个产品线，权限必须进行限制，防止跨产品造成意外。鉴于不希望经常请求多个产品线进行维护，各种不靠谱的自动升级机制层出不穷。由于运维成本大，开发者倾向于在一个客户端中塞进尽可能多的功能。
 - 出现自动升级机制
 - 客户端开始和操作系统一起安装
 - 存在各类权限控制方案
 - 每当客户端崩溃的时候，求root密码

- ✓ 代表作：NOAH Client 1.1
- ✓ 由于监控内容太多太杂，于是提供了OP自写脚本来扩充监控范围的方法，每台服务器的监控方法从此变得不同。然后发现这个东西（跑一个脚本）可以替代ssh做很多事，于是又加入了接受远程指令，从指定位置下载文件并且执行之的功能。虽然不知道应该怎么自动升级，但还是强行写了一个不怎么work的。总出故障。其（糟糕的）设计思想影响了后面一大堆客户端。
- ✓ 部署方法：rpm随装机安装。



- ✓ 日益增多的客户端，运维难度越来越大，于是包管理的概念开始出现。客户端需要遵守统一的规范（端口、监控方法、文件布局、control脚本等），并经由专门的客户端管理系统进行维护。由于运维难度降低，这是一个客户端繁荣的时代，不写个自己的客户端都不好意思和人打招呼。
 - 不再有安装、升级、启停等功能
 - 客户端的发布只需要一个标准包
 - 存在各类权限控制方案
 - 每当客户端崩溃的时候，求客户端组支持



- ✓ 代表作：Apollo
- ✓ 高鲁棒性的客户端管理系统
 - 根据机型、发行版来部署兼容的客户端
 - 接管了客户端的部署、升级、自动修复等工作
 - 实时统计部署率



- ✓ 高度通用的客户端开始逐渐被专用的客户端排挤。客户端的数量开始爆发，发行版的概念开始出现。专用的客户端中，职责相同、功能相似、理念矛盾的孩子们开始打架。
- ✓ 发行版的矛盾最终变得不可调和，即使是设计为通用型的客户端，也越来越难以应付各种不兼容问题。
 - 客户端的功能分化
 - 打架的客户端开始变多
 - 权限控制问题成了心腹大患
 - 中心管理机构无法承担太重的协调工作



- ✓ 代表作：正在做！
- ✓ 虚拟化是唯一的解决方案，但是现存的虚拟化方案，不管是轻还是重，都不能拿来即用。



客户端的背景

百度客户端发展史



客户端的设计原则



- ✓ 客户端是一个很渺小的东西，每当大牛们讨论牛逼的自动化系统的时候，所有人的视线都集中在分布式一致性、文件系统、负载均衡、P2P、failover等高级架构上。
- ✓ 然后人们才发现没有手脚，那些都是浮云。



- ✓ 客户端没有所谓的思想、架构、guideline
- ✓ 客户端只是有一大堆的、小小的技术点



- ✓ Rule No.1
- ✓ 能不写客户端就不要写客户端
 - 主流发行版经过20年以上的发展，任何单机管理工具都已经有了内建/开源的实现方案，也许不那么和你的需求贴切，但是基本上都能通过搭积木来work



- ✓ Rule No.1
- ✓ 能不写客户端就不要写客户端
 - 需要执行个命令？ssh
 - 需要定时执行个命令？cron
 - 需要上传/下载个文件？scp、rsync
 - 需要打日志？syslogd
 - 需要远程管理？ipmi、snmp
 - 需要监控健康状况？sysstat



- ✓ Rule No.2
- ✓ 能不起守护进程就不要起守护进程
 - Daemon要做好一跑就是半年的准备，再小的失误，一个月之后都会庞大到难以想象。
 - Daemon通常运行在高特权上，编程失误可能导致的损害、系统的被攻击面都比普通程序大。

看完APUE就信心满满要写daemon的孩子，一般都要搞砸。



✓ Rule No.2

- ✓ 能不起守护进程就不要起守护进程
 - 是不是可以做成工具（库），由业务主动调用？
 - 是不是可以做成业务的伴随服务，随着业务控制脚本启停？
 - 是不是可以依附于其他稳定的客户端来实现麻烦的部分？（比如sshd、crond、xinetd）
 - 是不是可以system(3)一个setuid的程序来代替？



✓ Rule No.3

✓ 能不用root就不要用root（包括setuid）

- 栽两次跟头大家都明白了。
- 只有代表操作系统的意志时，才能使用root。
- 任何机器帐号（daemon、ftp、nobody.....）都是root的一部分，它们代表了操作系统的一部分意志。



- ✓ Rule No.4
- ✓ 能不切换用户身份就不要切换用户身份
 - 一旦遇到需要代表业务帐号做点什么的时候，通俗地讲：su，就超级麻烦了。
 - system(3)一个/bin/su，比绝大部分自己实现的都要靠谱。



- ✓ 一句话guidelines
 - 不管硬件发展如何迅猛，客户端只能占用极少的资源
 - 一跑就是半年，一宕机就是一大片
 - 单机重试可靠性无敌，集群重试就是DDoS
 - 发现设计外的异常，别挣扎，直接crash
 - And a lot more...



客户端的背景

百度客户端发展史

客户端的设计原则

几个典型的坑



✓ 自升级系统

- 客户端能够迭代的基础
- 什么功能都可以没有，先把升级问题搞定
- 外部升级
 - ssh上去装（烦）
- 内部升级
 - 自己下载新版本，替换，重启自己（难）
- 外部升级
 - 统一的babysitter系统负责升级，客户端不再有自升级系统

- ✓ 什么都会fail
 - 做假设之前想一想fallback
 - 默认shell是nologin ? !
 - 没有diff命令 ? !
 - inode满了 ? !



✓ 跨平台支持

- 百度95%的服务器环境是同质的
- 即使是这样，我们还是踩到了大量的坑
 - 发行版bug的规避
 - 不同发行版的差异
 - 发行版的行为发生变更



- ✓ And a lot more...
- ✓ 客户端技术是大量的经验与教训的集合
- ✓ 最被忽视的组件，但是难度却是极大的

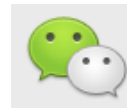


www.infoq.com/cn

InfoQ^{ueue}



@InfoQ



infoqchina

软件
正在改变世界!