LINUX PID 1 和 SYSTEMD

2017年07月16日 (Https://Coolshell.Cn/Articles/17998.Html) 陈皓 (Https://Coolshell.Cn/Articles/Author/Haoel)

要说清 Systemd，得先从Linux操作系统的启动说起。Linux 操作系统的启动首先从 BIOS 开始，然后由 Boot Loader 载入内核，并初始化内核。内核初始化的最后一步就是启动 init 进程。这个进程是系统的第一个进程，PID 为 1，又叫超级进程，也叫根进程。它负责产生其他所有用户进程。所有的进程都会被挂在这个进程下，如果这个进程退出了，那么所有的进程都被 kill 。如果一个子进程的父进程退了，那么这个子进程会被挂到 PID 1 下面。（注：PID 0 是内核的一部分，主要用于内进换页，参看：Process identifier (http://en.wikipedia.org/wiki/Process_identifier) ）

SysV Init

PID 1 这个进程非常特殊，其主要就任务是把整个操作系统带入可操作的状态。比如：启动 UI – Shell 以便进行人机交互，或者进入 X 图形窗口。传统上，PID 1 和传统的 Unix System V 相兼容的，所以也叫 `sysvinit` ，这是使用得最悠久的 init 实现。Unix System V 于1983年 release。

在 `sysvint` 下，有好几个运行模式，又叫 `runlevel` 。比如：常见的 3 级别指定启动到多用户的字符命令行界面，5 级别指定启起到图形界面，0 表示关机，6 表示重启。其配置在 `/etc/inittab` 文件中。

与此配套的还有 `/etc/init.d/` 和 `/etc/rc[X].d` ，前者存放各种进程的启停脚本（需要按照规范支持 `start` ，`stop` 子命令），后者的 X 表示不同的 runlevel 下相应的后台进程服务，如： `/etc/rc3.d` 是 runlevel=3 的。 里面的文件主要是 link 到 `/etc/init.d/` 里的启停脚本。其中也有一定的命名规范：S 或 K 打头的，后面跟一个数字，然后再跟一个自定义的名字，如： `S01rsyslog` ， `S02ssh` 。S 表示启动，K表示停止，数字表示执行的顺序。

UpStart

Unix 和 Linux 在 `sysvint` 运作多年后，大约到了2006年的时候，Linux内核进入2.6时代，Linux有了很多更新。并且，Linux开始进入桌面系统，而桌面系统和服务器系统不一样的是，桌面系统面临频繁重启，而且，用户会非常频繁的使用硬件的热插拔技术。于是，这些新的场景，让 `sysvint` 受到了很多挑战。

比如，打印机需要CUPS等服务进程，但是如果用户没有打机印，启动这个服务完全是一种浪费，而如果不启动，如果要用打印机了，就无法使用，因为 `sysvint` 没有自动检测的机制，它只能一次性启动所有的服务。另外，还有网络盘挂载的问题。在 `/etc/fstab` 中，负责硬盘挂载，有时候还有网络硬盘（NFS 或 iSCSI）在其中，但是在桌面机上，有很可能开机的时候是没有网络的， 于是网络硬盘都不可以访问，也无法挂载，这会极大的影响启动速度。 `sysvinit` 采用 `netdev` 的方式来解决这个问题，也就是说，需要用户自己在 `/etc/fstab` 中给相应的硬盘配置上 `netdev` 属性，于是 `sysvint` 启动时不会挂载它，只有在网络可用后，由专门的 `netfs` 服务进程来挂载。这种管理方式比较难以管理，也很容易让人掉坑。

所以，Ubuntu 开发人员在评估了当时几个可选的 init 系统后，决定重新设计这个系统，于是，这就是我们后面看到的 `upstart` 。 `upstart` 基于事件驱动的机制，把之前的完全串行的同步启动服务的方式改成了由事件驱动的异步的方式。比如：如果有U盘插入， `udev` 得到通知， `upstart` 感知到这个事件后触发相应的服务程序，比如挂载文件系统等等。因为使用一个事件驱动的玩法，所以，启动操作系统时，很多不必要的服务可以不用启动，而是等待通知，lazy 启动。而且事件驱动的好处是，可以并行启动服务，他们之间的依赖关系，由相应的事件通知完成。

upstart 有着很不错的设计，其中最重要的两个概念是 Job 和 Event。

Job 有一般的Job，也有service的Job，并且， `upstart` 管理了整个 Job 的生命周期，比如：Waiting, Starting, pre-Start, Spawned, post-Start, Running, pre-Stop, Stopping, Killed, post-Stop等等，并维护着这个生命周期的状态机。

Event 分成三类，`signal，method 和 hooks`。 `signal` 就是异步消息， `method` 是同步阻塞的。 `hooks` 也是同步的，但介于前面两者之间，发出hook事件的进程必须等到事件完成，但不检查是否成功。

但是， `upstart` 的事件非常复杂，也非常纷乱，各种各样的事件（事件没有归好类）导致有点凌乱。不过因为整个事件驱动的设计比之前的 `sysvinit` 来说好太多，所以，也深得欢迎。

Systemd

直到2010的有一天，一个在 RedHat 工作的工程师 Lennart Poettering (https://en.wikipedia.org/wiki/Lennart_Poettering) 和 Kay Sievers (https://en.wikipedia.org/wiki/Kay_Sievers)，开始引入了一个新的 `init` 系统—— `systemd`。这是一个非常非常有野心的项目，这个项目几乎改变了所有的东西， `systemd` 不但想取代已有的 init 系统，而且还想干更多的东西。

Lennart 同意 `upstart` 干的不错，代码质量很好，基于事件的设计也很好。但是他觉得 `upstart` 也有问题，其中最大的问题还是不够快，虽然 `upstart` 用事件可以达到一定的启动并行度，但是，本质上来说，这些事件还是会让启动过程串行在一起。 如： `NetworkManager` 在等 `D-Bus` 的启动事件，而 `D-Bus` 在等 `syslog` 的启动事件。

Lennart 认为，实现上来说， `upstart` 其实是在管理一个逻辑上的服务依赖树，但是这个服务依赖树在表现形式上比较简单，你只需要配置——“启动 B好了就启动A”或是“停止了A后就停止B”这样的规则。但是，Lennart 说，这种简单其实是有害的（this simplification is actually detrimental）。他认为，

- 从一个系统管理的角度出来，他一开始会设定好整个系统启动的服务依赖树，但是这个系统管理员要人肉的把这个本来就非常干净的服务依整树给翻译成计算机看的懂的 Event/Action 形式，而且 Event/Action 这种配置方式是运行时的，所以，你需要运行起来才知道是什么样的。
什么是运行时的？
- Event逻辑从头到脚到处都是，这个事件扩大了运维的复杂度，还不如之前的 `sysvint`。 也就是说，当用户配置了 “启动 `D-Bus` 后请启动 `NetworkManager`” ，这个 `upstart` 可以干，但是反过来，如果，用户启动 `NetworkManager` ，我们应该先去启动他的前置依赖 `D-Bus` ，然而你还要配置相应的反向 Event。本来，我只需要配置一条依赖的，结果现在我要配置很多很多情况下的Event。
- 最后， `upstart` 里的 Event 的并不标准，很混乱，没有良好的定义。比如：既有，进程启动，运行，停止的事件，也有USB设备插入、可用、拔出的事件，还有文件系统设备being mounted、 mounted 和 umounted 的事件，还有AC电源线连接和断开的事件。你会发现，这进程启停的、USB的、文件系统的、电源线的事件，看上去长得很像， 但是没有被标准化抽像出来掉，因为绝大多数的事件都是三元组：start, condition, stop 。这种概念设计模型并没有在 `upstart` 中出现。因为 `upstart` 被设计为单一的事件，而忽略了逻辑依赖。

当然，如果 `systemd` 只是解决 `upstart` 的问题，他就改造 `upstart` 就好了，但是 Lennart 的野心不只是想干个这样的事，他想干的更多。

首先， `systemd` 清醒的认识到了 init 进程的首要目标是要让用户快速的进入可以操作OS的环境，所以，这个速度一定要快，越快越好，所以， `systemd` 的设计理念就是两条：

- To start **less**.
- And to start **more** in *parallel*.

也就是说，按需启动，能不启动就不启动，如果要启动，能并行启动就并行启动，包括你们之间有依赖，我也并行启动。按需启动还好理解，那么，有依赖关系的并行启动，它是怎么做到的？这里， `systemd` 借鉴了 MacOS 的 `Launchd` 的玩法（在Youtube上有一个分享——Launchd: One Program to Rule them All (https://www.youtube.com/watch?v=SjrtySM9Dns)，在苹果的开源网站上也有相关的设计文档——About Daemons and Services (https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/Introduction.html)）

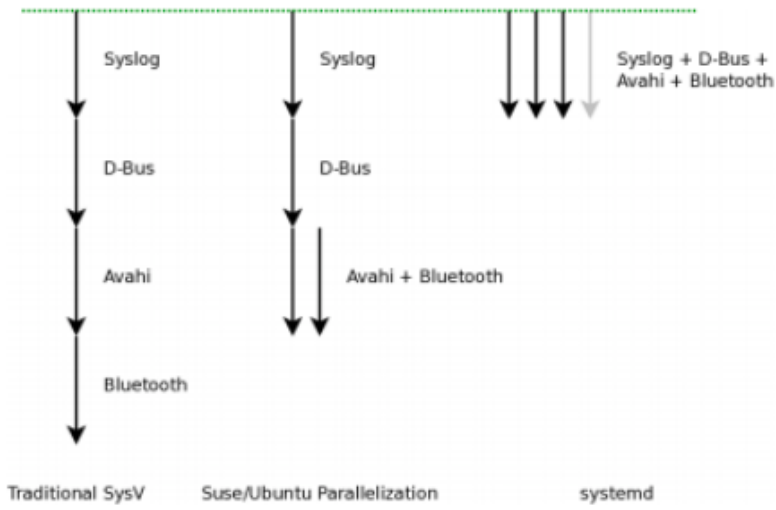
要解决这些依赖性，systemd 需要解决好三种底层依赖—— Socket， D-Bus ，文件系统。

- **Socket依赖。**如果服务C依赖于服务S的socket，那么就要先启动S，然后再启动C，因为如果C启动时找不到S的Socket，那么C就会失败。 `systemd` 可以帮你在S还没有启动好的时候，建立一个socket，用来接收所有的C的请求和数据，并缓存之，一旦S全部启动完成，把systemd替换好的这个缓存的数据和Socket描述符替换过去。

了解什么是D-Bus
- **D-Bus依赖。** `D-Bus` 全称 Desktop Bus，是一个用来在进程间通信的服务。除了用于用户态进程和内核态进程通信，也用于用户态的进程之前。现在，很多的现在的服务进程都用 `D-Bus` 而不是Socket来通信。比如： `NetworkManager` 就是通过 `D-Bus` 和其它服务进程通讯的，也就是说，如果一个进程需要知道网络的状态，那么就必需要通过 `D-Bus` 通信。 `D-Bus` 支持 “Bus Activation”的特性。也就是说，A要通过 `D-Bus` 服务和B通讯，但是B没有启动，那么 `D-Bus` 可以把B起来，在B启动的过程中， `D-Bus` 帮你缓存数据。 `systemd` 可以帮你利用好这个特性来并行启动 A 和 B。

- **文件系统依赖。**系统启动过程中，文件系统相关的活动是最耗时的，比如挂载文件系统，对文件系统进行磁盘检查（fsck），磁盘配额检查等都是非常耗时的操作。在等待这些工作完成的同时，系统处于空闲状态。那些想使用文件系统的服务似乎必须等待文件系统初始化完成才可以启动。 `systemd` 参考了 `autofs` 的设计思路，使得依赖文件系统的服务和文件系统本身初始化两者可以并发工作。 `autofs` 可以监测到某个文件系统挂载点真正被访问到的时候才触发挂载操作，这是通过内核 `automounter` 模块的支持而实现的。比如一个 `open()` 系统调用作用在某个文件系统上的时候，而这个文件系统尚未执行挂载，此时 `open()` 调用被内核挂起等待，等到挂载完成后，控制权返回给 `open()` 系统调用，并正常打开文件。这个过程和 `autofs` 是相似的。

下图来自 Lennart 的演讲里的一页PPT，展示了不同 init 系统的启动。



除此之外，systemd 还在启动时管理好了一些下面的事。

用C语言取代传统的脚本式的启动。前面说过， `sysvint` 用 `/etc/rcX.d` 下的各种脚本启动。然而这些脚本中需要使用 `awk`，`sed`，`grep`，`find`，`xargs` 等等这些操作系统的命令，这些命令需要生成进程，生成进程的开销很大，关键是生成完这些进程后，这个进程就干了点屁大的事就退了。换句话说就是，我操作系统干了那么多事为你拉个进程起来，结果你就把个字串转成小写就退了，把我操作系统当什么了？

在正常的一个 `sysvinit` 的脚本里，可能会有成百上千个这样的命令。所以，慢死。因此， `systemd` 全面用 C 语言全部取代了。一般来说， `sysvinit` 下，操作系统启动完成后，用 `echo $$` 可以看到，pid 被分配到了上千的样子，而 `systemd` 的系统只是上百。

另外，systemd 是真正一个可以管住服务进程的——可以跟踪上服务进程所fork/exec出来的所有进程。

- 我们知道， 传统 Unix/Linux 的 Daemon 服务进程的最佳实践基本上是这个样子的 （具体过程可参看这篇文章“SysV Daemon (http://0pointer.de/public/systemd-man/daemon.html#SysV%20Daemons)” ）——
  1. 进程启动时，关闭所有的打开的文件描述符（除了标准描述符0,1,2），然后重置所有的信号处理。
  2. 调用 `fork()` 创建子进程，在子进程中 `setsid()`，然后父进程退出（为了后台执行）为什么要关闭打开的文件描述符？关闭的描述符是如何打开的？
  3. 在子进程中，再调用一次 `fork()` ，创建孙子进程，确定没有交互终端。然后子进程退出。
  4. 在孙子进程中，把标准输入标准输出标准错误都连到 `/dev/null` 上，还要创建 pid 文件，日志文件，处理相关信号 ……
  5. 最后才是真正开始提供服务。

- 在上面的这个过程中，服务进程除了两次 `fork` 外还会 `fork` 出很多很多的子进程（比如说一些Web服务进程，会根据用户的请求链接来 `fork` 子进程），这个进程树是相当难以管理的，因为，一旦父进程退出来了，子进程就会被挂到 PID 1下，所以，基本上来说，你无法通过服务进程自己给定的一个pid文件来找到所有的相关进程（这个对开发者的要求太高了），所以，在传统的方式下用脚本启停服务是相当相当的 Buggy 的，因为无法做对所有的服务生出来的子子孙孙做到监控。

- 为了解决这个问题， `upstart` 通过变态的 `strace` 来跟踪进程中的 `fork()` 和 `exec()` 或 `exit()` 等相关的系统调用。这种方法相当笨拙。 `systemd` 使用了一个非常有意思的玩法来 tracking 服务进程生出来的所有进程，那就是用 `cgroup` （我在 Docker 的基础技术“cgroup 篇”(https://coolshell.cn/articles/17049.html)中讲过这个东西）。cgroup主要是用来管理进程组资源配额的事，所以，无论服务如何启动新的子进程，所有的这些相关进程都会同属于一个 `cgroup` ，所以， `systemd` 只需要简单的去遍历一下相应的 `cgroup` 的那个虚文件系统目录下的文件，就可以正确的找到所有的相关进程，并将他们一一停止。

另外， `systemd` 简化了整个 daemon 开发的过程：

- 不需要两次 `fork()` ，只需要实现服务本身的主逻辑就可以了。
- 不需要 `setsid()` ， `systemd` 会帮你干
- 不需要维护 `pid文件` ， `systemd` 会帮处理。
- 不需要管理日志文件或是使用 `syslog` ，或是处理 `HUP` 的日志reload信号。把日志打到 `stderr` 上， `systemd` 帮你管理。
- 处理 `SIGTERM` 信号，这个信号就是正确退出当前服务，不要做其他的事。
- ……

除此之外， `systemd` 还能——

- 自动检测启动的服务间有没有环形依赖。
- 内建 autofs 自动挂载管理功能。
- 日志服务。 `systemd` 改造了传统的 syslog 的问题，采用二进制格式保存日志，日志索引更快。
- 快照和恢复。对当前的系统运行的服务集合做快照，并可以恢复。
- ……

还有好多好多，他接管很多很多东西，于是就让很多人不爽了，因为他在干了很多本不属于 PID 1 的事。

Systemd 争论和八卦

于是 systemd 这个东西成了可能是有史以来口水战最多的一个开源软件了。 systemd 饱受各种争议，最大的争议就是他破坏了 Unix 的设计哲学（相关的哲学可以读一下《Unix 编程艺术(https://book.douban.com/subject/1467587/)》），干了一个大而全而且相当复杂的东西。当然，Lennart 并不同意这样的说法，他后来又写一篇 blog “The Biggest Myths (http://0pointer.de/blog/projects/the-biggest-myths.html)” 来解释 systemd 并不是这样的，大家可以前往一读。

这个争议大到什么样子呢？2014 年，Debian Linux 因为想准备使用 systemd 来作为标准的 init 守护进程来替换 sysvinit 。而围绕这个事的争论达到了空前的热度，争论中充满着仇恨， systemd 的支持者和反对者都在互相辱骂，导致当时 Debian 阵营开始分裂。还有人给 Lennart 发了死亡威胁的邮件，用比特币雇凶买杀手，扬言要取他的性命，在Youbute上传了侮辱他的歌曲，在IRC和各种社交渠道上给他发下流和侮辱性的消息。这已经不是争议了，而是一种不折不扣的仇恨！

于是，Lennart 在 Google Plus 上发了贴子(https://plus.google.com/+LennartPoetteringTheOneAndOnly/posts/J2TZrTvu7vd)，批评整个 Linux 开源社区和 Linus 本人。他大意说，

> *这个社区太病态了，全是 ass holes，你们不停用各种手段在各种地方用不同的语言和方式来侮辱和漫骂我。我还是一个年轻人，我从来没有经历过这样的场面，但是今天我已经对这种场面很熟悉了。我有时候说话可能不准确，但是我不会像他样那样说出那样的话，我也没有被这些事影响，因为我脸皮够厚，所以，为什么我可以在如何大的反对声面前让 systemd 成功，但是，你们 Linux 社区太可怕了。你们里面的有精神病的人太多了。另外，对于Linus Torvalds，你是这个社区的 Role Model，但可惜你是一个 Bad Role Model，你在社区里的刻薄和侮辱性的言行，基本从一定程度上鼓励了其它人跟你一样，当然，并不只是你一个人的问题，而是在你周围聚集了一群和你一样的这样干的人。送你一句话—— A fish rots from the head down ！一条鱼是从头往下腐烂的……*

这篇契文很长，喜欢八卦的同学可以前往一读。感受一下 Lennart 当时的心态（我觉得能算上是非常平稳了）。

Linus也在被一媒体问起 systemd 这个事来（参看 “Torvalds says he has no strong opinions on systemd (https://www.itwire.com/business-it-news/open-source/65402-torvalds-says-he-has-no-strong-opinions-on-systemd)” ），Linus在采访里说，

> *我对 systemd 和 Lennart 的贴子没有什么强烈的想法。虽然，传统的 Unix 设计哲学—— “Do one thing and Do it well” ，很不错，而且我们大多数人也实践了这么多年，但是这并不代表所有的真实世界。在历史上，也不只有 systemd 这么干过。但是，我个人还是 old-fashioned 的人，至少我喜欢文本式的日志，而不是二进制的日志。但是 systemd 没有必要一定要有这样的品味。哦，我说细节了……*

今天， systemd 占据了几乎所有的主流的 Linux 发行版的默认配置，包括：Arch Linux、CentOS、CoreOS、Debian、Fedora、Megeia、OpenSUSE、RHEL、SUSE企业版和 Ubuntu。而且，对于 CentOS, CoreOS, Fedora, RHEL, SUSE这些发行版来说，不能没有 systemd 。（Ubuntu 还有一个不错的wiki – Systemd for Upstart Users (https://wiki.ubuntu.com/SystemdForUpstartUsers) 阐述了如何在两者间切换）

其它

还记得在《缓存更新的套路 (https://coolshell.cn/articles/17416.html)》一文中，我说过，**如果你要做好架构，首先你得把计算机体系结构以及很多老古董的基础技术吃透了。**因为里面会有很多可以借鉴和相通的东西。那么，你是否从这篇文章里看到了一些有分布式架构相似的东西？

比如：从 `sysvinit` 到 `upstart` 再到 `systemd` ，像不像是服务治理？Linux系统下的这些服务进程，是不是很像分布式架构中的微服务？还有那个D-Bus，是不是很像SOA里的ESB？而 init 系统是不是很像一个控制系统？甚至像一个服务编排（Service Orchestration）系统？

分布式系统中的服务之间也有很多依赖，所以，在启动一个架构的时候，如果我们可以做到像 systemd 那样并行启动的话，那么是不是就像是一个微服务的玩法了？

嗯，你会发现，技术上的很多东西是相通的，也是互相有对方的影子，所以，其实技术并不多。关键是我们学在了表面还是看到了本质。

延伸阅读

- Lennert 的博文：Rethinking PID 1 (http://0pointer.de/blog/projects/systemd.html)
- Lennert 的演讲：systemd, beyond init (https://www.youtube.com/watch?v=TyMLi8QF6sw)（PPT (http://www.linux-kongress.org/2010/slides/systemd-poettering.pdf)）
- Wikipedia：Systemd (https://en.wikipedia.org/wiki/Systemd)
- LinuxVoice：Lennart Poettering 专访 (https://www.linuxvoice.com/interview-lennart-poettering/)

（全文完）