**systemd简述**

来自：http://blog.csdn.net/smstong/article/details/39317491

# 1、systemd是什么

首先systmed是一个用户空间的程序，属于应用程序，不属于Linux内核范畴，Linux内核的主要特征在所有发行版中是统一的，厂商可以自由改变的是用户空间的应用程序。

Linux内核加载启动后，用户空间的第一个进程就是初始化进程，这个程序的物理文件约定位于/sbin/init，当然也可以通过传递内核参数来让内核启动指定的程序。这个进程的特点是进程号为1，代表第一个运行的用户空间进程。不同发行版采用了不同的启动程序，主要有以下几种主流选择：

（1）以Ubuntu为代表的Linux发行版采用upstart。

（2）以7.0版本之前的CentOS为代表的System V init。

（3）CentOS7.0版本的systemd。

# 2、 Systemd物理文件组成

systemd是一个完整的软件包，安装完成后有很多物理文件组成，大致分布为，配置文件位于/etc/systemd这个目录下，配置工具命令位于/bin，和/sbin这两个目录下，预先准备的备用配置文件位于/lib/systemd目录下，还有库文件和帮助手册等等。这是一个庞大的软件包。详情使用rpm -ql systemd即可查看。

先让我们看看当前系统/etc/inittab这个文件的内容，这个文件是systme V init的标准配置文件，如今变成了:

1. # inittab is no longer used when using systemd.
2. #
3. # ADDING CONFIGURATION HERE WILL HAVE NO EFFECT ON YOUR SYSTEM.
4. #
5. # Ctrl-Alt-Delete is handled by /etc/systemd/system/ctrl-alt-del.target
6. #
7. # systemd uses 'targets' instead of runlevels. By default, there are two main targets:
8. #
9. # multi-user.target: analogous to runlevel 3
10. # graphical.target: analogous to runlevel 5
11. #
12. # To set a default target, run:
13. #
14. # ln -sf /lib/systemd/system/<target name>.target /etc/systemd/system/default.target

这说明，在systemd掌权后，inittab不再起作用，也没有了“运行级”的概念。现在起作用的配置文件是/etc/systemd/system/default.target这个文件了。

1. #  This file is part of systemd.
2. #
3. #  systemd is free software; you can redistribute it and/or modify it
4. #  under the terms of the GNU Lesser General Public License as published by
5. #  the Free Software Foundation; either version 2.1 of the License, or
6. #  (at your option) any later version.
8. [Unit]
9. Description=Multi-User System
10. Documentation=man:systemd.special(7)
11. Requires=basic.target
12. Conflicts=rescue.service rescue.target
13. After=basic.target rescue.service rescue.target
14. AllowIsolate=yes
16. [Install]
17. Alias=default.target

# 3、Systemd运行原理

## 3.1 systemd的基本概念

### （1）配置单元unit

系统初始化要做很多工作，如挂在文件系统，启动sshd服务，配置交换分区，这都可以看做是一个配置单元，systemd安装功能不同把配置单元分成多种类型。

service 后代服务进程，如httpd，mysqld等

soket 对应一个套接字，之后对应到一个service，类似于xinetd的功能

device 对应udev规则标记的一个设备

mount 系统中的一个挂载点，systemd据此进行自动挂载，为了与SystemV兼容，目前systemd自动处理/etc/fstab并转化为mount

automount 自动挂载点

swap 配置交换分区

target 配置单元的逻辑分组，包含多个相关的配置单元，可以当成是SystemV中的运行级。

timer 定时器。用来定时触发用户定义的操作，它可以用来取代传统的atd，crond等。

snapshot 与target类似，表示当前的运行状态

每一个配置单元都有一个对应的配置文件，系统管理员的任务就是编写和维护这写不同的配置文件，比如一个MySql服务对应一个mysql.service文件。

### （2）依赖关系

systemd并不能完全解除各个单元之间的依赖关系，如物理设备单元准备就绪之前，不可能执行挂载单元。为此需要定义各个单元之间的依赖关系。有依赖的地方就会有出现死循环的可能，比如A依赖于B，B依赖于C，C依赖于A，那么导致死锁。systemd为此提供了两种不同程度的依赖关系，一个是require，一个是want，出现死循环时，systemd会尝试忽略want类型的依赖，如仍不能解锁，那么systemd报错。

### （3)Target和runlevel

前面说过，systemd使用target取代了systemV的运行级的概念。

表 1. Sysvinit 运行级别和 systemd 目标的对应表

Sysvinit 运行级别 Systemd 目标 备注

0 runlevel0.target, poweroff.target 关闭系统。

1, s, single runlevel1.target, rescue.target 单用户模式。

2, 4 runlevel2.target, runlevel4.target, multi-user.target 用户定义/域特定运行级别。默认等同于 3。

3 runlevel3.target, multi-user.target 多用户，非图形化。用户可以通过多个控制台或网络登录。

5 runlevel5.target, graphical.target 多用户，图形化。通常为所有运行级别 3 的服务外加图形化登录。

6 runlevel6.target, reboot.target 重启

emergency emergency.target 紧急 Shell

## 3.2 systemd并行启动原理

如前所述，在 Systemd 中，所有的服务都并发启动，比如 Avahi、D-Bus、livirtd、X11、HAL 可以同时启动。乍一看，这似乎有点儿问题，比如 Avahi 需要 syslog 的服务，Avahi 和 syslog 同时启动，假设 Avahi 的启动比较快，所以 syslog 还没有准备好，可是 Avahi 又需要记录日志，这岂不是会出现问题？

Systemd 的开发人员仔细研究了服务之间相互依赖的本质问题，发现所谓依赖可以分为三个具体的类型，而每一个类型实际上都可以通过相应的技术解除依赖关系。

并发启动原理之一：解决 socket 依赖

绝大多数的服务依赖是套接字依赖。比如服务 A 通过一个套接字端口 S1 提供自己的服务，其他的服务如果需要服务 A，则需要连接 S1。因此如果服务 A 尚未启动，S1 就不存在，其他的服务就会得到启动错误。所以传统地，人们需要先启动服务 A，等待它进入就绪状态，再启动其他需要它的服务。Systemd 认为，只要我们预先把 S1 建立好，那么其他所有的服务就可以同时启动而无需等待服务 A 来创建 S1 了。如果服务 A 尚未启动，那么其他进程向 S1 发送的服务请求实际上会被 Linux 操作系统缓存，其他进程会在这个请求的地方等待。一旦服务 A 启动就绪，就可以立即处理缓存的请求，一切都开始正常运行。

那么服务如何使用由 init 进程创建的套接字呢？

Linux 操作系统有一个特性，当进程调用 fork 或者 exec 创建子进程之后，所有在父进程中被打开的文件句柄 (file descriptor) 都被子进程所继承。套接字也是一种文件句柄，进程 A 可以创建一个套接字，此后当进程 A 调用 exec 启动一个新的子进程时，只要确保该套接字的 close\_on\_exec 标志位被清空，那么新的子进程就可以继承这个套接字。子进程看到的套接字和父进程创建的套接字是同一个系统套接字，就仿佛这个套接字是子进程自己创建的一样，没有任何区别。

这个特性以前被一个叫做 inetd 的系统服务所利用。Inetd 进程会负责监控一些常用套接字端口，比如 Telnet，当该端口有连接请求时，inetd 才启动 telnetd 进程，并把有连接的套接字传递给新的 telnetd 进程进行处理。这样，当系统没有 telnet 客户端连接时，就不需要启动 telnetd 进程。Inetd 可以代理很多的网络服务，这样就可以节约很多的系统负载和内存资源，只有当有真正的连接请求时才启动相应服务，并把套接字传递给相应的服务进程。

和 inetd 类似，systemd 是所有其他进程的父进程，它可以先建立所有需要的套接字，然后在调用 exec 的时候将该套接字传递给新的服务进程，而新进程直接使用该套接字进行服务即可。

并发启动原理之二：解决 D-Bus 依赖

D-Bus 是 desktop-bus 的简称，是一个低延迟、低开销、高可用性的进程间通信机制。它越来越多地用于应用程序之间通信，也用于应用程序和操作系统内核之间的通信。很多现代的服务进程都使用D-Bus 取代套接字作为进程间通信机制，对外提供服务。比如简化 Linux 网络配置的 NetworkManager 服务就使用 D-Bus 和其他的应用程序或者服务进行交互：邮件客户端软件 evolution 可以通过 D-Bus 从 NetworkManager 服务获取网络状态的改变，以便做出相应的处理。

D-Bus 支持所谓"bus activation"功能。如果服务 A 需要使用服务 B 的 D-Bus 服务，而服务 B 并没有运行，则 D-Bus 可以在服务 A 请求服务 B 的 D-Bus 时自动启动服务 B。而服务 A 发出的请求会被 D-Bus 缓存，服务 A 会等待服务 B 启动就绪。利用这个特性，依赖 D-Bus 的服务就可以实现并行启动。

并发启动原理之三：解决文件系统依赖

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。但是 systemd 发现这种依赖也是可以避免的。

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"/misc/cd/file1"的时候，/misc/cd 尚未执行挂载操作，此时 open()调用被挂起等待，Linux 内核通知 autofs，autofs 执行挂载。这时候，控制权返回给 open()系统调用，并正常打开文件。

Systemd 集成了 autofs 的实现，对于系统中的挂载点，比如/home，当系统启动的时候，systemd 为其创建一个临时的自动挂载点。在这个时刻/home 真正的挂载设备尚未启动好，真正的挂载操作还没有执行，文件系统检测也还没有完成。可是那些依赖该目录的进程已经可以并发启动，他们的 open()操作被内建在 systemd 中的 autofs 捕获，将该 open()调用挂起（可中断睡眠状态）。然后等待真正的挂载操作完成，文件系统检测也完成后，systemd 将该自动挂载点替换为真正的挂载点，并让 open()调用返回。由此，实现了那些依赖于文件系统的服务和文件系统本身同时并发启动。

当然对于"/"根目录的依赖实际上一定还是要串行执行，因为 systemd 自己也存放在/之下，必须等待系统根目录挂载检查好。

不过对于类似/home 等挂载点，这种并发可以提高系统的启动速度，尤其是当/home 是远程的 NFS 节点，或者是加密盘等，需要耗费较长的时间才可以准备就绪的情况下，因为并发启动，这段时间内，系统并不是完全无事可做，而是可以利用这段空余时间做更多的启动进程的事情，总的来说就缩短了系统启动时间。

# 4 、Systemd配置使用

## 4.1 对于系统开发人员

开发人员需要了解 systemd 的更多细节。比如您打算开发一个新的系统服务，就必须了解如何让这个服务能够被 systemd 管理。这需要您注意以下这些要点：

后台服务进程代码不需要执行两次派生来实现后台精灵进程，只需要实现服务本身的主循环即可。

不要调用 setsid()，交给 systemd 处理

不再需要维护 pid 文件。

Systemd 提供了日志功能，服务进程只需要输出到 stderr 即可，无需使用 syslog。

处理信号 SIGTERM，这个信号的唯一正确作用就是停止当前服务，不要做其他的事情。

SIGHUP 信号的作用是重启服务。

需要套接字的服务，不要自己创建套接字，让 systemd 传入套接字。

使用 sd\_notify()函数通知 systemd 服务自己的状态改变。一般地，当服务初始化结束，进入服务就绪状态时，可以调用它。

对于开发者来说，工作量最大的部分应该是编写配置单元文件，定义所需要的单元。

举例来说，开发人员开发了一个新的服务程序，比如 httpd，就需要为其编写一个配置单元文件以便该服务可以被 systemd 管理，类似 UpStart 的工作配置文件。在该文件中定义服务启动的命令行语法，以及和其他服务的依赖关系等。

此外我们之前已经了解到，systemd 的功能繁多，不仅用来管理服务，还可以管理挂载点，定义定时任务等。这些工作都是由编辑相应的配置单元文件完成的。我在这里给出几个配置单元文件的例子。

下面是 SSH 服务的配置单元文件，服务配置单元文件以.service 为文件名后缀。

#cat /etc/system/system/sshd.service

[Unit]

Description=OpenSSH server daemon

[Service]

EnvironmentFile=/etc/sysconfig/sshd

ExecStartPre=/usr/sbin/sshd-keygen

ExecStart=/usrsbin/sshd –D $OPTIONS

ExecReload=/bin/kill –HUP $MAINPID

KillMode=process

Restart=on-failure

RestartSec=42s

[Install]

WantedBy=multi-user.target

文件分为三个小节。第一个是[Unit]部分，这里仅仅有一个描述信息。第二部分是 Service 定义，其中，ExecStartPre 定义启动服务之前应该运行的命令；ExecStart 定义启动服务的具体命令行语法。第三部分是[Install]，WangtedBy 表明这个服务是在多用户模式下所需要的。

那我们就来看下 multi-user.target 吧：

#cat multi-user.target

[Unit]

Description=Multi-User System

Documentation=man.systemd.special(7)

Requires=basic.target

Conflicts=rescue.service rescure.target

After=basic.target rescue.service rescue.target

AllowIsolate=yes

[Install]

Alias=default.target

第一部分中的 Requires 定义表明 multi-user.target 启动的时候 basic.target 也必须被启动；另外 basic.target 停止的时候，multi-user.target 也必须停止。如果您接着查看 basic.target 文件，会发现它又指定了 sysinit.target 等其他的单元必须随之启动。同样 sysinit.target 也会包含其他的单元。采用这样的层层链接的结构，最终所有需要支持多用户模式的组件服务都会被初始化启动好。

在[Install]小节中有 Alias 定义，即定义本单元的别名，这样在运行 systemctl 的时候就可以使用这个别名来引用本单元。这里的别名是 default.target，比 multi-user.target 要简单一些。。。

此外在/etc/systemd/system 目录下还可以看到诸如\*.wants 的目录，放在该目录下的配置单元文件等同于在[Unit]小节中的 wants 关键字，即本单元启动时，还需要启动这些单元。比如您可以简单地把您自己写的 foo.service 文件放入 multi-user.target.wants 目录下，这样每次都会被默认启动了。

最后，让我们来看看 sys-kernel-debug.mout 文件，这个文件定义了一个文件挂载点：

#cat sys-kernel-debug.mount

[Unit]

Description=Debug File Syste

DefaultDependencies=no

ConditionPathExists=/sys/kernel/debug

Before=sysinit.target

[Mount]

What=debugfs

Where=/sys/kernel/debug

Type=debugfs

这个配置单元文件定义了一个挂载点。挂载配置单元文件有一个[Mount]配置小节，里面配置了 What，Where 和 Type 三个数据项。这都是挂载命令所必须的，例子中的配置等同于下面这个挂载命令：

mount –t debugfs /sys/kernel/debug debugfs

配置单元文件的编写需要很多的学习，必须参考 systemd 附带的 man 等文档进行深入学习。希望通过上面几个小例子，大家已经了解配置单元文件的作用和一般写法了。

## 4.2 对于系统管理员

系统管理员的主要工具是systemctl。

多数管理员应该都已经非常熟悉系统服务和 init 系统的管理，比如 service、chkconfig 以及 telinit 命令的使用。systemd 也完成同样的管理任务，只是命令工具 systemctl 的语法有所不同而已，因此用表格来对比 systemctl 和传统的系统管理命令会非常清晰。

表 2. Systemd 命令和 sysvinit 命令的对照表

| **Sysvinit 命令** | **Systemd 命令** | **备注** |
| --- | --- | --- |
| service foo start | systemctl start foo.service | 用来启动一个服务 (并不会重启现有的) |
| service foo stop | systemctl stop foo.service | 用来停止一个服务 (并不会重启现有的)。 |
| service foo restart | systemctl restart foo.service | 用来停止并启动一个服务。 |
| service foo reload | systemctl reload foo.service | 当支持时，重新装载配置文件而不中断等待操作。 |
| service foo condrestart | systemctl condrestart foo.service | 如果服务正在运行那么重启它。 |
| service foo status | systemctl status foo.service | 汇报服务是否正在运行。 |
| ls /etc/rc.d/init.d/ | systemctl list-unit-files --type=service | 用来列出可以启动或停止的服务列表。 |
| chkconfig foo on | systemctl enable foo.service | 在下次启动时或满足其他触发条件时设置服务为启用 |
| chkconfig foo off | systemctl disable foo.service | 在下次启动时或满足其他触发条件时设置服务为禁用 |
| chkconfig foo | systemctl is-enabled foo.service | 用来检查一个服务在当前环境下被配置为启用还是禁用。 |
| chkconfig -list | systemctl list-unit-files --type=service | 输出在各个运行级别下服务的启用和禁用情况 |
| chkconfig foo -list | ls /etc/systemd/system/\*.wants/foo.service | 用来列出该服务在哪些运行级别下启用和禁用。 |
| chkconfig foo -add | systemctl daemon-reload | 当您创建新服务文件或者变更设置时使用。 |
| telinit 3 | systemctl isolate multi-user.target (OR systemctl isolate runlevel3.target OR telinit 3) | 改变至多用户运行级别。 |

除了表 2 列出的常见用法，系统管理员还需要了解其他一些系统配置和管理任务的改变。

首先我们了解 systemd 如何处理电源管理，命令如下表所示：

表 3，systemd 电源管理命令

命令 操作

systemctl reboot 重启机器

systemctl poweroff 关机

systemctl suspend 待机

systemctl hibernate 休眠

systemctl hybrid-sleep 混合休眠模式（同时休眠到硬盘并待机）

关机不是每个登录用户在任何情况下都可以执行的，一般只有管理员才可以关机。正常情况下系统不应该允许 SSH 远程登录的用户执行关机命令。否则其他用户正在工作，一个用户把系统关了就不好了。为了解决这个问题，传统的 Linux 系统使用 ConsoleKit 跟踪用户登录情况，并决定是否赋予其关机的权限。现在 ConsoleKit 已经被 systemd 的 logind 所替代。

logind 不是 pid-1 的 init 进程。它的作用和 UpStart 的 session init 类似，但功能要丰富很多，它能够管理几乎所有用户会话(session)相关的事情。logind 不仅是 ConsoleKit 的替代，它可以：

维护，跟踪会话和用户登录情况。如上所述，为了决定关机命令是否可行，系统需要了解当前用户登录情况，如果用户从 SSH 登录，不允许其执行关机命令；如果普通用户从本地登录，且该用户是系统中的唯一会话，则允许其执行关机命令；这些判断都需要 logind 维护所有的用户会话和登录情况。

Logind 也负责统计用户会话是否长时间没有操作，可以执行休眠/关机等相应操作。

为用户会话的所有进程创建 CGroup。这不仅方便统计所有用户会话的相关进程，也可以实现会话级别的系统资源控制。

负责电源管理的组合键处理，比如用户按下电源键，将系统切换至睡眠状态。

多席位(multi-seat) 管理。如今的电脑，即便一台笔记本电脑，也完全可以提供多人同时使用的计算能力。多席位就是一台电脑主机管理多个外设，比如两个屏幕和两个鼠标/键盘。席位一使用屏幕 1 和键盘 1；席位二使用屏幕 2 和键盘 2，但他们都共享一台主机。用户会话可以自由在多个席位之间切换。或者当插入新的键盘，屏幕等物理外设时，自动启动 gdm 用户登录界面等。所有这些都是多席位管理的内容。ConsoleKit 始终没有实现这个功能，systemd 的 logind 能够支持多席位。

以上描述的这些管理功能仅仅是 systemd 的部分功能，除此之外，systemd 还负责系统其他的管理配置，比如配置网络，Locale 管理，管理系统内核模块加载等。

# 5 总结

在不才作者看来，作为系统初始化系统，systemd 的最大特点有两个：

令人惊奇的激进的并发启动能力，极大地提高了系统启动速度；

用 CGroup 统计跟踪子进程，干净可靠。

此外，和其前任不同的地方在于，systemd 已经不仅仅是一个初始化系统了。

Systemd 出色地替代了 sysvinit 的所有功能，但它并未就此自满。因为 init 进程是系统所有进程的父进程这样的特殊性，systemd 非常适合提供曾经由其他服务提供的功能，比如定时任务 (以前由 crond 完成) ；会话管理 (以前由 ConsoleKit/PolKit 等管理) 。仅仅从本文皮毛一样的介绍来看，Systemd 已经管得很多了，可它还在不断发展。它将逐渐成为一个多功能的系统环境，能够处理非常多的系统管理任务，有人甚至将它看作一个操作系统。

好的一点是，这非常有助于标准化 Linux 的管理！从前，不同的 Linux 发行版各行其事，使用不同方法管理系统，从来也不会互相妥协。比如如何将系统进入休眠状态，不同的系统有不同的解决方案，即便是同一个 Linux 系统，也存在不同的方法，比如一个有趣的讨论：如何让 ubuntu 系统休眠，可以使用底层的/sys/power/state 接口，也可以使用诸如 pm-utility 等高层接口。存在这么多种不同的方法做一件事情对像我这样的普通用户而言可不是件有趣的事情。systemd 提供统一的电源管理命令接口，这件事情的意义就类似全世界的人都说统一的语言，我们再也不需要学习外语了，多么美好！

如果所有的 Linux 发行版都采纳了 systemd，那么系统管理任务便可以很大程度上实现标准化。此外 systemd 有个很棒的承诺：接口保持稳定，不会再轻易改动。对于软件开发人员来说，这是多么体贴又让人感动的承诺啊！