1.为什么是systemd

2017年2月13日　　12:49

（1） 关于Linux服务管理

Linux系统从启动到提供服务的过程是这样，先是机器加电，然后通过MBR或者UEFI加载GRUB，再启动内核，内核启动服务，然后开始对外服务。
SysV init UpStart systemd主要是解决服务引导管理的问题。
提示：关于systemd的拼写，官方的说法就是systemd，既不是Syetemd，也是不systemD。

（2） SysV init的优缺点

SysV init是最早的解决方案，依靠划分不同的运行级别，启动不同的服务集，服务依靠脚本控制，并且是顺序执行的。
SysV init方案的优点是：
原理简单，易于理解；
依靠shell脚本控制，编写服务脚本门槛比较低。
缺点是：
服务顺序启动，启动过程比较慢；
不能做到根据需要来启动服务，比如通常希望插入U盘的时候，再启动USB控制的服务，这样可以更好的节省系统资源。

（3） UpStart的改进

为了解决系统服务的即插即用，UpStart应运而生，在CentOS6系统中，SysV init和UpStart是并存的，UpStart主要解决了服务的即插即用。服务顺序启动慢的问题，UpStart的解决办法是把相关的服务分组，组内的服务是顺序启动，组之间是并行启动。

（4） systemd的诞生

SysV init服务启动慢，在以前并不是一个问题，尤其是Linux系统以前主要是在服务器系统上，常年也难得重启一次。有的服务器光硬件检测都需要5分钟以上，相对来说系统启动已经很快了。
但是随着移动互联网的到来，SysV init服务启动慢的问题显得越来越突出，许多移动设备都是基于Linux内核，比如安卓。移动设备启动比较频繁，每次启动都要等待服务顺序启动，显然难以接受，systemd就是为了解决这个问题诞生的。
systemd的设计思路是：
尽可能的快速启动服务；

尽可能的减少系统资源占用。

（5）为什么systemd能做到启动很快

systemd使用并行的方法启动服务，不像SysV init是顺序执行的，所以大大节省了系统启动时间。
使用并行启动，最大的难点是要解决服务之间的依赖性，systemd的解决办法是使用类似缓冲池的办法。比如对TCP有依赖的服务，在启动的时候会检查依赖服务的TCP端口，systemd会把对TCP端口的请求先缓存起来，当依赖的服务器启动之后，在将请求传递给服务，使两个服务通讯。同样的进程间通讯的D-BUS也是这样的原理，目录挂载则是先让服务以为目录被挂载了，到真正访问目录的时候，才去真正操作。

2.SysV init

2017年2月13日 12:50

2.SysV init介绍

SysV init是systemV风格的init系统，顾名思义，它源于SystemV系列UNIX。它提供了比BSD风格init系统更高的灵活性。是已经风行了几十年的UNIX init系统，一直被各类Linux发行版所采用。

（1）什么是SystemV

SystemV，曾经也被称为AT&T SystemV，是Unix操作系统众多版本中的一支。它最初由AT&T开发，在1983年第一次发布。一共发行了4个SystemV的主要版本：版本1、2、3和4。SystemV Release4，或者称为SVR4，是最成功的版本，成为一些UNIX共同特性的源头，例如”SysV初始化脚本“（/etc/init.d），用来控制系统启动和关闭，SystemV Interface Definition(SVID)是一个SystemV如何工作的标准定义。

（2）SysV init的运行级别

SysV init用术语runlevel来定义"预订的运行模式"。SysV init检查'/etc/inittab'文件中是否含有'initdefault'项。来告诉init系统是否有一个默认运行模式。如果没有默认的运行模式，那么用户将进入系统控制台，手动决定进入何种运行模式。

SysV init中运行模式描述了系统各种预订的运行模式。通常会有8种运行模式，即运行模式0到6和S或者s。

每种Linux发行版对运行模式的定义都不太一样。但0，1，6却得到了大家的一致赞同：

0关机

1单用户模式

6重启

通常在/etc/inittab文件中定义了各种运行模式的工作范围。比如RedHat定义了runlevel3和5。运行模式3将系统初始化为字符界面的shell模式；运行模式5将系统初始化为GUI模式。无论是命令行界面还是GUI，运行模式3和5相对于其他运行模式而言都是完整的正式的运行状态，计算机可以完成用户需要的任务。而模式1，S等往往用于系统故障之后的排错和恢复。

很显然，这些不同的运行模式下系统需要初始化运行的进程，需要进行的初始化准备都是不同的。比如运行模式3不需要启动X系统。用户只需要指定需要进入哪种模式，SysV init负责执行所有该模式所必须的初始化工作。

（3）SysV init运行顺序

SysV init巧妙地用脚本，文件命名规则和软链接来实现不同的runlevel。首先，SysV init需要读取/etc/inittab文件。分析这个文件的内容，它获得以下一些配置信息：
系统需要进入的runlevel；
捕获组合键的定义；
定义电源fail/restore脚本；
启动getty和虚拟控制台；
得到配置信息后，SysV init顺序地执行以下这些步骤，从而将系统初始化为预订的runlevelX：
/etc/rc.d/rc.sysinit
/etc/rc.d/rc和/etc/rc.d/rcX.d/(X代表运行级别0-6)
/etc/rc.d/rc.local
XDisplayManager（如果需要的话）

1）rc.sysinit脚本功能

首先，运行rc.sysinit以便执行一些重要的系统初始化任务。在RedHat公司的RHEL5中(RHEL6已经使用UpStart了)，rc.sysinit主要完成以下这些工作：
激活udev和selinux；
设置定义在/etc/sysctl.conf中的内核参数；
设置系统时钟；
加载keymaps；
激活交换分区；
设置主机名(hostname)；
根分区检查和remount；
激活RAID和LVM设备；
开启磁盘配额；
检查并挂载所有文件系统；
清除过期的locks和PID文件；

2）rc.d脚本

完成了以上这些工作之后，SysV init开始运行/etc/rc.d/rc脚本。根据不同的runlevel，rc脚本将打开对应runlevel的rcX.d目录(X就是runlevel)，找到并运行存放在该目录下的所有启动脚本。每个runlevelX都有一个这样的目录，目录名为/etc/rc.d/rcX.d。
在这些目录下存放着很多不同的脚本。文件名以S开头的脚本就是启动时应该运行的脚本，S后面跟的数字定义了这些脚本的执行顺序。
在/etc/rc.d/rcX.d目录下的脚本其实都是一些软链接文件，真实的脚本文件存放在/etc/init.d目录下。如下所示：
rc5.d目录下的脚本

[root@www~]#ll/etc/rc5.d/
lrwxrwxrwx1rootroot16Sep42008K02dhcdbd->../init.d/dhcdbd
....(中间省略)....
lrwxrwxrwx1rootroot14Sep42008K91capi->../init.d/capi
lrwxrwxrwx1rootroot23Sep42008S00microcode_ctl->../init.d/microcode_ctl
lrwxrwxrwx1rootroot22Sep42008S02lvm2-monitor->../init.d/lvm2-monitor
....(中间省略)....
lrwxrwxrwx1rootroot17Sep42008S10network->../init.d/network
....(中间省略)....
lrwxrwxrwx1rootroot11Sep42008S99local->../rc.local
lrwxrwxrwx1rootroot16Sep42008S99smartd->../init.d/smartd
....(底下省略)....

当所有的初始化脚本执行完毕。SysV init运行/etc/rc.d/rc.local脚本。
rc.local是Linux留给用户进行个性化设置的地方。可以把自己私人想设置和启动的东西放到这里，一台LinuxServer的用户一般不止一个，所以才有这样的考虑。

（4）SysV init和系统关闭

SysV init不仅需要负责初始化系统，还需要负责关闭系统。在系统关闭时，为了保证数据的一致性，需要小心地按顺序进行结束和清理工作。比如应该先停止对文件系统有读写操作的服务，然后再umount文件系统。否则数据就会丢失。
这种顺序的控制这也是依靠/etc/rc.d/rcX.d/目录下所有脚本的命名规则来控制的，在该目录下所有以K开头的脚本都将在关闭系统时调用，字母K之后的数字定义了它们的执行顺序。
这些脚本负责安全地停止服务或者其他的关闭工作。

（5）SysV init的管理和控制功能

此外，在系统启动之后，管理员还需要对已经启动的进程进行管理和控制。SysV init软件包包含了一系列的控制启动，运行和关闭所有其他程序的工具。
halt停止系统。
init就是SysV init本身的init进程实体，以pid1身份运行，是所有用户进程的父进程。最主要的作用是在启动过程中使用/etc/inittab文件创建进程。
killall5就是System V的killall命令。向除自己会话(session)进程之外的其它进程发出信号，所以不能杀死当前使用的shell。
last回溯/var/log/wtmp文件(或者-f选项指定的文件)，显示自从这个文件建立以来，所有用户的登录情况。
lastb作用和last差不多，默认情况下使用/var/log/btmp文件，显示所有失

败登录企图。
mesg控制其它用户对用户终端的访问。
pidof找出程序的进程识别号(pid)，输出到标准输出设备。
poweroff等于shutdown-h–p，或者telinit0。关闭系统并切断电源。
reboot等于shutdown–r或者telinit6。重启系统。
runlevel读取系统的登录记录文件(一般是/var/run/utmp)把以前和当前的系统运行级输出到标准输出设备。
shutdown以一种安全的方式终止系统，所有正在登录的用户都会收到系统将要终止通知，并且不准新的登录。
sulogin当系统进入单用户模式时，被init调用。当接收到启动加载程序传递的-b选项时，init也会调用sulogin。
telinit实际是init的一个连接，用来向init传送单字符参数和信号。
utmpdump以一种用户友好的格式向标准输出设备显示/var/run/utmp文件的内容。
wall向所有有信息权限的登录用户发送消息。
不同的Linux发行版在这些SysV init的基本工具基础上又开发了一些辅助工具用来简化init系统的管理工作。比如RedHat的RHEL在SysV init的基础上开发了initscripts软件包，包含了大量的启动脚本(如rc.sysinit)，还提供了service，chkconfig等命令行工具，甚至一套图形化界面来管理init系统。其他的Linux发行版也有各自的initscript或其他名字的init软件包来简化SysV init的管理。
只要理解了SysV init的机制，在一个最简的仅有SysV init的系统下，可以直接调用脚本启动和停止服务，手动创建inittab和创建软连接来完成这些任务。因此理解SysV init的基本原理和命令是最重要的。甚至也可以开发自己的一套管理工具。

3.systemd的特性

2017年2月13日　　12:50

1）systemd解决了那些问题？

按需启动服务，减少系统资源消耗；
尽可能并行启动进程，减少系统启动等待时间；
提供一个一致的配置环境，不光是服务配置；
提供服务状态快照，可以恢复特定点的服务状态。

（2）systemd的争议在哪里？

systemd试图提供一个一致的配置环境，请注意不光是服务配置，还包括其他方面的系统配置，这个也是systemd的野心，希望能把Linux系统上的配置统一起来，为了达到这个目标，systemd牺牲掉了SysV init和BSD系统的兼容性。systemd充分利用了Linux内核API，不在支持BSD系统，这个是systemd目前在开源社区最大的争论。
个人人为这个是个好事情，对管理员来讲，再也不用学习不同发行版上不同的配置，也不用为了写一个脚本，先去写一堆判断，判断不同的发型版。运维自动化的前提是标准化，只有标准化了，才能自动化，systemd朝这个方向走了一大步。即使systemd的学习成本比较高。

（3）systemd能更彻底的结束服务进程

服务(daemon)进程，为了成为服务，会fork两次，所以进程编号会发生变化，UpStart在结束进程的时候，有可能会找错进程编号，造成服务永远不被停止。
还有更特殊的情况，如果进程产生了子进程，子进程又自己fork了两次，脱离了主进程，要结束这样的子进程，UpStart的办法是通过strace追踪fork，exit调用，这种方法非常复杂。
systemd利用了内核最新的特性，使用CGroup解决这个问题，CGroup的进程是树状的，因此无论服务如何启动新的子进程，所有的这些相关进程都会属于同一个CGroup，systemd只需要简单地遍历指定的CGroup即可正确地找到所有的相关进程，将它们一一停止即可。

4.CentOS 7的systemd特性

2017年2月13日 12:50

4.CentOS 7的systemd特性

（1）套接字服务保持激活功能

在系统启动的时候，systemd为所有支持套接字激活功能的服务创建监听端口，当服务启动后，就将套接字传给这些服务。这种方式不仅可以允许服务在启动的时候平行启动，也可以保证在服务重启期间，试图连接服务的请求，不会丢失。对服务端口的请求被保留，并且存放到队列中。

（2）进程间通讯保持激活功能

当有客户端应用第一次通过D-Bus方式请求进程间通讯时，systemd会立即启动对应的服务。systemd依据D-Bus的配置文件使用进程间通讯保持激活功能。

（3）设备保持激活功能

当特定的硬件插入时，systemd启动对应的硬件服务支持。systemd依据硬件服务单元配置文件保持硬件随时被激活。

（4）文件路径保持激活功能

当特定的文件或者路径状态发生改变的时候，systemd会激活对应的服务。systemd依据路径服务单元配置文件保证服务被激活。

（5）系统状态快照

systemd可以临时保存当前所有的单元配置文件，或者从前一个快照中恢复单元配置文件。为了保存当前系统服务状态，systemd可以动态的生成单元文件快照。

（6）挂载和自动挂载点管理

systemd监控和管理挂载和自动挂载点，并根据挂载点的单元配置文件进行挂载。

（7）闪电并行启动

因为使用套接字保持激活功能，systemd可以并行的启动所以套接字监听

服务，大大减少系统启动时间。

（8）单元逻辑模拟检查

当激活或者关闭一个单元，systemd会计算依赖行，产生一个临时的模拟检查，并且校验一直性。如果不一致，systemd会尝试自动修正，并且移除报错的不重要的任务。

（9）和SysV init向后兼容

systemd完全支持SysV initLinux标准的基础核心规范脚本，这样的脚本易于升级到systemd服务单元。

5.如何分析衡量systemd启动速度

2017年2月13日　　12:50

5.如何分析衡量systemd启动速度

systemd-analyze是一个分析启动性能的工具，用于分析启动时服务时间消耗。默认显示启动是内核和用户空间的消耗时间：

[root@localhost~]#systemd-analyze
Startupfinishedin818ms(kernel)+6.240s(initrd)+32.979s(userspace)=40.038s

和使用systemd-analyzetime命令的效果一样。

（1）查看详细的每个服务消耗的启动时间

通过systemd-analyzeblame命令查看详细的每个服务消耗的启动时间：

[root@localhost~]#systemd-analyzeblame
30.852siscsi.service
16.994skdump.service
10.871sboot.mount
...
103mssystemd-sysctl.service
101msdatapool.mount

（2）查看严重消耗时间的服务树状表

systemd-analyzecritical-chain命令打印严重消耗时间的服务树状表，按照启动消耗的时间进行排序，时间消耗越多，越排到前面。@之后是服务激活或者启动的时间，+号之后是服务启动消耗的时间。个人理解@是从系统引导到服务启动起来的时间，是一个相对时间消耗，+是服务启动消耗的时间，是一个绝对时间消耗。

[root@localhost~]#systemd-analyzecritical-chain
Thetimeaftertheunitisactiveorstartedisprintedafterthe"@"character.
Thetimetheunittakestostartisprintedafterthe"+"character.
multi-user.target@32.976s
└─kdump.service@15.981s+16.994s
└─network.target@15.980s
└─NetworkManager.service@15.069s+54ms
└─firewalld.service@14.532s+535ms
└─basic.target@14.532s

└─sockets.target@14.532s
└─dbus.socket@14.532s
└─sysinit.target@14.527s
└─systemd-update-utmp.service@14.524s+2ms
└─systemd-tmpfiles-setup.service@14.456s+67ms
└─local-fs.target@14.447s
└─boot.mount@3.575s+10.871s
└─systemd-fsck@dev-disk-by\x2duuid-8c77568b\x2d7e51\x2d4e32\x2dbbdf\x2ddc12ff737bbf.service@3.348s+226ms
└─systemd-fsck-root.service@1.237s+152ms
└─systemd-readahead-replay.service@1.073s+25ms

（3）打印分析图及其他命令

systemd-analyzeplot打印一个svg格式的服务消耗时间表，通过浏览器可以以图形的方式展示，非常直观：

[root@localhost~]#systemd-analyzeplot>plot.svg

CentOS7/RHEL7 systemd详解

其他参数：
systemd-analyzedot用分隔符产生当前服务
systemd-analyzedump以友好方式显示当前服务状态
6systemd文件类型及存放位置
systemd配置文件被称为unit单元，根据类型不同，以不同的扩展名结尾。
.service系统服务；
.target一组系统服务；
.automount自动挂载点；
.device能被内核识别的设备；
.mount挂载点；
.path文件系统的文件或者目录；
.scope外部创建的进程；
.slice一组分层次管理的系统进程；
.snapshot系统服务状态管理；
.socket进程间通讯套接字；
.swap定义swap文件或者设备；
.timer定义定时器。

6.CentOS 7的systemd向后兼容

2017年2月13日 12:50

6.CentOS 7的systemd向后兼容

systemd被设计成尽可能向后兼容SysV init和Upstart，下面是一些特别要注意的和之前主要版本的RHEL不再兼容的部分。

（1）systemd对运行级别支持有限。

为了保存兼容，systemd提供一定数量的target单元，可以直接和运行级别对应，也可以被早期的分布式的运行级别命令支持。不是所有的target都可以被映射到运行级别，在这种情况下，使用runlevel命令有可能会返回一个为N的不知道的运行级别，所以推荐尽量避免在RHEL7中使用runlevel命令。

（2）systemd不支持像init脚本那样的个性化命令。

除了一些标准命令参数例如：start、stop、status，SysV init脚本可以根据需要支持想要的任何参数，通过参数提供附加的功能，因为SysV init的服务器脚本实际上就是shell脚本，命令参数实际上就是shell子函数。举个例子，RHEL6的iptables服务脚本可以执行panic命令行参数，这个参数可以让系统立即进入紧急模式，丢弃所有的进入和发出的数据包。但是类似这样的命令行参数在systemd中是不支持的，systemd只支持在配置文件中指定命令行参数。

（3）systemd不支持和没有从systemd启动的服务通讯。

当systemd启动服务的时候，他保存进程的主ID以便于追踪，systemctl工具使用进程PID查询和管理服务。相反的，如果用户从命令行启动特定的服务，systemctl命令是没有办法判断这个服务的状态是启动还是运行的。

（4）systemd可以只停止运行的服务

在RHEL6及之前的版本，当关闭系统的程序启动之后，RHEL6的系统会执行/etc/rc0.d/下所有服务脚本的关闭操作，不管服务是处于运行或者根本没有运行的状态。而systemd可以做到只关闭在运行的服务，这样可以大大节省关机的时间。

（5）不能从标准输出设备读到系统服务信息。

systemd启动服务的时候，将标准输出信息定向到/dev/null，以免打扰用

户。

（6）systemd不继承任何上下文环境。

systemd不继承任何上下文环境，如用户或者会话的HOME或者PATH的环境变量。每个服务得到的是干净的上下文环境。

（7）SysV init脚本依赖性

当systemd启动SysV init脚本，systemd在运行的时候，从LinuxStandardBase(LSB)Linux标准库头文件读取服务的依赖信息并继承。

（8）超时机制

为了防止系统被卡住，所有的服务有5分钟的超时机制。

7.systemd服务管理

2017年2月13日　　12:51

7.systemd服务管理

（1）什么是单元

在RHEL7之前，服务管理是分布式的被SysV init或UpStart通过/etc/rc.d/init.d下的脚本管理。这些脚本是经典的Bash脚本，允许管理员控制服务的状态。在RHEL7中，这些脚本被服务单元文件替换。
在systemd中，服务、挂载等资源统一被称为单元，所以systemd中有许多单元类型，服务单元文件的扩展名是.service，同脚本的功能相似。例如有查看、启动、停止、重启、启用或者禁止服务的参数。
systemd单元文件放置位置：
/usr/lib/systemd/system/systemd默认单元文件安装目录
/run/systemd/systemsystemdsystemd单元运行时创建，这个目录优先于按照目录
/etc/systemd/system系统管理员创建和管理的单元目录，优先级最高。

（2）systemd的服务管理

使用systemcl命令可以控制服务，service命令和chkconfig命令依然可以使用，但是主要是出于兼容的原因，应该尽量避免使用。
使用systemctl命令的时候，服务名字的扩展名可以写全，例如：
systemctl stop bluuetooth.service
也可以忽略，例如：
systemctl stop bluetooth
systemctl常用命令：
启动服务 systemctl start name.service
关闭服务 systemctl stop name.service
重启服务 systemctl restar tname.service
仅当服务运行的时候，重启服务 systemctl try-restart name.service
重新加载服务配置文件 systemctl relaod name.service
检查服务运作状态 systemctl status name.service 或者 systemctl is-active\
name.service
展示所有服务状态详细信息 systemctl list-units--type service --all
允许服务开机启动 systemctl enable name.service
禁止服务开机启动 systemclt disable name.service
检查服务开机启动状态 systemctl status name.service 或者systemctl\
is-enabled name.service
列出所有服务并且检查是否开机启动 systemctl list-unit-files --type service

（3）服务详细信息查看

使用如下命令列出服务：
systemctl list-units --type service
默认只列出处于激活状态的服务，如果希望看到所有的服务，使用--all或-a参数：
systemctl list-units--type service --all
有时候希望看到所以可以设置开机启动的服务，使用如下命令：
systemctl list-unit-files --type service
查看服务详细信息，使用如下命令：
systemctl status name.service
服务信息关键词解释
Loaded服务已经被加载，显示单元文件绝对路径，标志单元文件可用。
Active服务已经被运行，并且有启动时间信息。
Main PID与进程名字一致的PID，主进程PID。
Status服务的附件信息。
Process相关进程的附件信息。
CGroup进程的CGroup信息。

8.使用systemd target

2017年2月13日　　12:51

8.使用systemd target

（1）怎样知道一个目标需要哪些进程服务？

例如，可能想搞明白目标单元multi-user.target究竟启用了哪些服务，使用以下命令：

$systemctlshow-p"Wants"multi-user.target
Wants=rc-local.serviceavahi-daemon.servicerpcbind.serviceNetworkManager.serviceacpid.servicedbus.serviceatd.servicecrond.serviceauditd.servicentpd.serviceudisks.servicebluetooth.serviceorg.cups.cupsd.servicewpa_supplicant.servicegetty.targetmodem-manager.serviceportreserve.serviceabrtd.serviceyum-updatesd.serviceupowerd.servicetest-first.servicepcscd.servicersyslog.servicehaldaemon.serviceremote-fs.targetplymouth-quit.servicesystemd-update-utmp-runlevel.servicesendmail.servicelvm2-monitor.servicecpuspeed.serviceudev-post.servicemdmonitor.serviceiscsid.servicelivesys.servicelivesys-late.serviceirqbalance.serviceiscsi.service

除了Wants，还可以查看各种形式的依赖和被依赖信息：WantedBy、Requires、RequiredBy、Conflicts、ConflictedBy、Before、After。

（2）target与运行级别

在RHEL7之前的版本，使用运行级别代表特定的操作模式。运行级别被定义为七个级别，用数字0到6表示，每个级别可以启动特定的一些服务。RHEL7使用target替换运行基本。
systemd target使用target单元文件描述，target单位文件扩展名是.target，target单元文件的唯一目标是将其他systemd单元文件通过一连串的依赖关系组织在一起。举个例子，graphical.target单元，用于启动一个图形会话，systemd会启动像GNOME显示管理(gdm.service)、帐号服务（axxounts-daemon）这样的服务，并且会激活multi-user.target单元。相似的multi-user.target单元，会启动必不可少的NetworkManager.service、dbus.service服务，并激活basic.target单元。
RHEL7预定义了一些target和之前的运行级别或多或少有些不同。为了兼容，systemd也提供一些target映射为SysV init的运行级别，具体的对应信息如下：

0runlevel0.target,poweroff.target关闭系统。
1runlevel1.target,rescue.target进入救援模式。
2runlevel2.target,multi-user.target进入非图形界面的多用户方式。
3runlevel3.target,multi-user.target进入非图形界面的多用户方式。
4runlevel4.target,multi-user.target进入非图形界面的多用户方式。
5runlevel5.target,graphical.target进入图形界面的多用户方式。
6runlevel6.target,reboot.target重启系统。

（3）target管理

1）使用如下命令查看目前可用的target：

systemctl list-units --type target

改变当前的运行基本使用如下命令：

1 systemctl isolate name.target

2）修改默认的运行级别
使用systemctl get-default命令得到默认的运行级别：

[root@localhost~]#systemctlget-default
multi-user.target

使用systemctl set-default name.target修改默认的运行基本

[root@localhost~]#systemctlset-defaultgraphical.target
rm'/etc/systemd/system/default.target'
ln-
s'/usr/lib/systemd/system/graphical.target''/etc/systemd/system/default.target'

3）救援模式和紧急模式
使用systemctl rescue进入救援模式，如果连救援模式都进入不了，可以进入紧急模式：

systtmctl emergency

紧急模式进入做小的系统环境，以便于修复系统。紧急模式根目录以只读方式挂载，不激活网络，只启动很少的服务，进入紧急模式需要root密码。

9.关闭、暂停、休眠系统

2017年2月13日　　12:51

9.关闭、暂停、休眠系统

RHEL7中，使用systemctl替换一些列的电源管理命令，原有的命令依旧可以使用，但是建议尽量不用使用。systemctl和这些命令的对应关系为：
hatl，systemctl halt停止系统
poweroff，systemctl poweroff关闭系统，关闭系统电源。
reboot，systemctl reboot重启系统
pm-suspend，systemctl suspend暂停系统
pm-hibernate，systemct lhibernate休眠系统
pm-suspend-hybrid，systemctl hybrid-sleep暂停并休眠系统

10.通过systemd管理远程系统

2017年2月13日　　12:51

10.通过systemd管理远程系统

不光是可以管理本地系统，systemd还可以控制远程系统，管理远程系统主要是通过SSH协议，只有确认可以连接远程系统的SSH，在systemctl命令后面添加-H或者--host参数，加上远程系统的ip或者主机名就可以。

11.创建和修改systemd单元文件

2017年2月13日　　12:51

11.创建和修改systemd单元文件

（1）单元文件概述

单元文件包含单元的指令和行为信息。在后台systemctl命令和单元文件一起工作。为了出色而正确的完成工作，系统管理员必须能够手工编辑单元文件。一般系统管理员手工创建的单元文件建议存放
在/etc/systemd/system/目录下面。
单元配置文件的格式是：

unit_name.type_extension

这里的unit_name代表单元名称，type_extension代表单元类型。
单元文件可以作为附加的文件放置到一个目录下面，比如为了定制sshd.service服务，可以创建sshd.service.d/custom.conf文件，在文件中做一些自定义的配置。
同样的，可以创建sshd.service.wants/和sshd.service.requires/目录。这些目录包含sshd服务关联服务的软连接，在系统安装的时候，这些软连接或自动创建，也可以手工创建软连接。
许多单元配置文件可以使用单元说明符--通配的字符串，可以在单元文件被引导的时候动态的被变量替换。这使创建一些通用的单元配置模版成为可能。

（2）理解单元文件结构

典型的单元文件包含三节：
[Unit]节，包含不依赖单元类型的一般选项，这些选型提供单元描述，知道单元行为，配置单元和其他单元的依赖性。
[unittype]节，如果单元有特定的类型指令，在unittype节这些指令被组织在一起。举个例子，服务单元文件包含[Service]节，里面有经常使用的服务配置。
[Install]节，包含systemctlenable或者disable的命令安装信息。
1）[Unit]节选项
Description单元描述信息，这些文字信息在systemclstatus命令是会输出。
Documentation单元文档信息的URLs。
After定义在那些单元之后启动，本单元只在制定的单元启动之后启动，不像Requires选项，After选项不明确激活特定的单元，Before选项则是有相反的功能。

Requires配置单元的依赖性，在Requires选项中的单元需要一起被激活，如果有一个单元启动失败，其他单元都不会被启动。
Wants比Requires选项依赖性要弱很多，如果列表之中的的单元启动失败，不会对其他单元造成影响，这是推荐的建立自定义单元依赖性的方式。
Conflicts定义单元冲突关系，和Requires相反。
2）[unittype]类型是[Service]时的选项
Type配置单元进程在启动时候的类型，影响执行和关联选项的功能，可选的关键字是：
simple默认值，进程和服务的主进程一起启动；
forking进程作为服务主进程的一个子进程启动，父进程在完全启动之后退出。
oneshot同simple相似，但是进程在启动单元之后随之退出。
dbus同simple相似，但是随着单元启动后只有主进程得到D-BUS名字。
notify同simple相似，但是随着单元启动之后，一个主要信息被sd_notify()函数送出。
idle同simple相似，实际执行进程的二进制程序会被延缓直到所有的单元的任务完成，主要是避免服务状态和shell混合输出。
ExecStart指定启动单元的命令或者脚本，ExecStartPre和ExecStartPost节指定在ExecStart之前或者之后用户自定义执行的脚本。Type=oneshot允许指定多个希望顺序执行的用户自定义命令。
ExecStop指定单元停止时执行的命令或者脚本。
ExecReload指定单元重新加载是执行的命令或者脚本。
Restart这个选项如果被允许，服务重启的时候进程会退出，会通过systemctl命令执行清除并重启的操作。
RemainAfterExit如果设置这个选择为真，服务会被认为是在激活状态，即使所以的进程已经退出，默认的值为假，这个选项只有在Type=oneshot时需要被配置。
3）[Install]节选项
Alias为单元提供一个空间分离的附加名字。
RequiredBy单元被允许运行需要的一系列依赖单元，RequiredBy列表从Require获得依赖信息。
WantBy单元被允许运行需要的弱依赖性单元，Wantby从Want列表获得依赖信息。
Also指出和单元一起安装或者被协助的单元。
DefaultInstance实例单元的限制，这个选项指定如果单元被允许运行默认的实例。
4）一个postfix服务的例子：
单元文件位于/usr/lib/systemd/system/postifix.service，内容如下：

```
[Unit]
Description=PostfixMailTransportAgent
After=syslog.targetnetwork.target
```

```
Conflicts=sendmail.serviceexim.service
[Service]
Type=forking
PIDFile=/var/spool/postfix/pid/master.pid
EnvironmentFile=-/etc/sysconfig/network
ExecStartPre=-/usr/libexec/postfix/aliasesdb
ExecStartPre=-/usr/libexec/postfix/chroot-update
ExecStart=/usr/sbin/postfixstart
ExecReload=/usr/sbin/postfixreload
ExecStop=/usr/sbin/postfixstop
[Install]
WantedBy=multi-user.target
```

（3）创建自定义的单元文件

以下几种场景需要自定义单元文件：
希望自己创建守护进程；
为现有的服务创建第二个实例；
引入SysV init脚本。
另外一方面，有时候需要修改已有的单元文件。
下面介绍创建单元文件的步骤：
1）准备自定义服务的执行文件。
可执行文件可以是脚本，也可以是软件提供者的的程序，如果需要，为自定义服务的主进程准备一个PID文件，一保证PID保持不变。另外还可能需要的配置环境变量的脚本，确保所以脚本都有可执行属性并且不需要交互。
2）在/etc/systemd/system/目录创建单元文件，并且保证只能被root用户编辑：

```
touch/etc/systemd/system/name.servicechmod664/etc/systemd/system/name.service
```

文件不需要执行权限。
3）打开name.service文件，添加服务配置，各种变量如何配置视所添加的服务类型而定，下面是一个依赖网络服务的配置例子：

```
[Unit]
Description=service_description
After=network.target
[Service]
ExecStart=path_to_executable
Type=forking
PIDFile=path_to_pidfile
```

[Install]
WantedBy=default.target

4）通知systemd有个新服务添加：

systemctldaemon-reload
systemctlstartname.service

（4）创建emacs.service例子：

1）创建文件，并确保正确权限：

~]#touch/etc/systemd/system/emacs.service
~]#chmod664/etc/systemd/system/emacs.service

2）添加配置信息：

[Unit]
Description=Emacs:theextensible,self-documentingtexteditor
[Service]
Type=forking
ExecStart=/usr/bin/emacs--daemon
ExecStop=/usr/bin/emacsclient--eval"(kill-emacs)"
Environment=SSH_AUTH_SOCK=%t/keyring/ssh
Restart=always
[Install]
WantedBy=default.target

3）通知systemd并开启服务：

~]#systemctldaemon-reload
~]#systemctlstartemacs.service

（5）创建第二个sshd服务的例子

1）拷贝sshd_config文件

]#cp/etc/ssh/sshd{,-second}_config

2）编辑sshd-second_config文件，添加22220的端口，和PID文件：

Port22220
PidFile/var/run/sshd-second.pid

如果还需要修改其他参数，请阅读帮助。
3）拷贝单元文件：

~]#cp/usr/lib/systemd/system/sshd{,-second}.service

4）编辑单元文件sshd-second.service
修改描述字段
Description=OpenSSHserversecondinstancedaemon
添加sshd.service服务在After关键字之后：
After=syslog.targetnetwork.targetauditd.servicesshd.service
移除sshdkey创建：
ExecStartPre=/usr/sbin/sshd-keygen移除这一行
在执行脚本里，添加第二sshd服务的配置文件：
ExecStart=/usr/sbin/sshd-D-f/etc/ssh/sshd-second_config$OPTIONS
修改后的sshd-second.service文件内容如下：

[Unit]
Description=OpenSSHserversecondinstancedaemon
After=syslog.target network.targe tauditd.service sshd.service
[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D -f /etc/ssh/sshd-second_config$OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s
[Install]
WantedBy=multi-user.target

5）如果使用SELinux，添加tcp端口，负责第二sshd服务的端口就会被拒绝绑定：

~]#semanage port -a -tssh_port_t -p tcp22220

6）设置开机启动并测试：

12 ~]#systemctl enable sshd-second.service
~]$ssh -p 22220 user@server

确保防火墙端口也开放。

（6）修改已经存在的单元文件

systemd单元配置文件默认保存在/usr/lib/systemd/system/目录，系统管理员不建议直接修改这个目录下的文件，自定义的文件
在/etc/systemd/system/目录下，如果有扩展的需求，可以使用以下方案：
创建一个目录/etc/systemd/system/unit.d/，这个是最推荐的一种方式，可以参考初始的单元文件，通过附件配置文件来扩展默认的配置，对默认单元文件的升级会被自动升级和应用。
从/usr/lib/systemd/system/拷贝一份原始配置文件
到/etc/systemd/system/，然后修改。复制的版本会覆盖原始配置，这种方式不能增加附件的配置包，用于不需要附加功能的场景。
如果需要恢复到默认的配置文件，只需要删除/etc/systemd/system/下的配置文件就可以了，不需要重启机器，使用如下命令应用改变就可以：

systemctl daemon-reload

daemon-reload选项重新加载所以单元文件并重新创建依赖书，在需要立即应用单元文件改变的时候使用。另外，也可以使用下面的命令达到同样的目的：

init q

还有，如果修改的是一个正在运行服务的单元文件，服务需要被重启下：

systemct lrestart name.service

（7）扩展默认单元配置文件配置

为了扩展默认的单元文件配置，需要先在/etc/systemd/system/下创建一个目录，用root执行类似下面的命令：

mkdir/etc/systemd/system/name.service.d

在刚才创建的目录之下创建配置文件，必须以.conf文件结尾。
例如创建一个自定义的依赖文件，内容如下：

[Unit]
Requires=new_dependency
After=new_dependency

另外一个例子，可以配置重启的时候，在主进程退出后30秒在重启，配置例子如下：

[Service]
Restart=always

RestartSec=30

推荐每次只产生一个小文件，每个文件只聚焦完善一个功能，这样配置文件很容易被移除或者链接到其他服务对的配置目录中。
为了应用刚才的修改，使用root执行以下操作：

systemctldaemon-reload
systemctlrestartname.service

例子：扩展httpd.service服务配置
为了是httpd服务启动的时候执行用户自定义的脚本，需要修改httpd的单元配置文件，执行以下几步操作，首先创建一个自定义文件的目录及自定义文件：

~]#mkdir/etc/systemd/system/httpd.service.d
~]#touch/etc/systemd/system/httpd.service.d/custom_script.conf

假设自定义文件位置在/usr/local/bin/custom.sh，将这个信息添加到custom_script.conf自定义脚本中：

[Service]
ExecStartPost=/usr/local/bin/custom.sh

应用更改：

~]#systemctldaemon-reload
~]#systemctlrestarthttpd.service

12.单元实例化

2017年2月13日　　12:51

在运行的时候有可能需要将一个模版实例化好几个单元，@字符用于标识模版和单元文件的关系，实例化单元可以从另外一个单元文件（使用Requires或者Wants选项），或者使用systemctlstart命令。实例化服务单元可以按照下面的方式命名：

template_name@instance_name.service

几个实例可以指向同一个模板文件配置选项常见的所有实例，举个例子，一个单元配置文件的Wants选项可以是：

Wants=getty@ttyA.service,getty@ttyB.service

首先让systemd搜索给定服务单位，如果没有发现，systemd忽略@和点号之间的部分，直接搜索getty@.service服务文件，读取配置，并启动服务。通配符字段，称为单元说明符，可以在任何单元配置文件使用。单位说明符替代某些单位在运行时参数和解释。常用的单元说明符说明如下：
%n整个单元名字，包括类型的后缀，%N是相同的意义，但是ASCII取代为禁止字符。
%p前缀名字，在实例化的时候，%p代表@字符前面的部分。
%i实例名字，@字符和单元类型直接的部分。%I是相同的意义，但是ASCII取代为禁止字符。
%H主机名字，当配置文件被加载的时候的主机名。
%t运行时目录，当前的运行目录，对root用户就是/run目录，对于无特权用户就是XDG_RUNTIME_DIR变量指定的目录。
举个例子，getty@.service包含下面的结构：

```
[Unit]
Description=Gettyon%I
...
[Service]
ExecStart=-/sbin/agetty--noclear%I$TERM
...
```

当getty@ttyA.service和getty@ttyB.service实例化的时候，Description=被解释为“GettyonttyA”和"GettyonttyB"。

13.VNC SERVER配置

2017年2月13日　　12:51

13.VNC SERVER配置

安装：

yum install tigervnc-server

配置：

（1） 复制配置文件：

~]# cp /lib/systemd/system/vncserver@.service \
/etc/systemd/system/vncserver@.service

（2） 编辑配置文件：

ExecStart=/sbin/runuser -l USER -c "/usr/bin/vncserver %i -geometry 1280x1024"
PIDFile=/home/USER/.vnc/%H%i.pid

将USER换成要使用的VNC服务的用户，比如root:

ExecStart=/sbin/runuser -l root -c "/usr/bin/vncserver %i"

如果要修改分辨率可以修改geometry内容，其他不需要做修改。
然后保持配置。

（3）使用systemctl命令，强制重新读取配置文件：

~]# systemctl daemon-reload

（4）配置vncserver密码

vncpasswd

（5）如果有两个用户希望同时使用vnc，需要配置两份配置文件：
vncserver-USER_1@.service 及 vncserver-USER_2@.service，文件内容同root用户的配置方法
然后为两个用户创建vnc密码：

$ su - USER_1
~]$ vncpasswd
Password:
Verify:
~]$ su - USER_2

```
~]$ vncpasswd
Password:
Verify:
```

（6）启动vnc服务

```
systemctl start vncserver@:10
```

为了开机启动，使用如下命令：

```
systemctl enable vncserver@:10
ln -s '/etc/systemd/system/vncserver@.service' \
'/etc/systemd/system/multi-user.target.wants/vncserver@:10.service'
```

（7） 关闭进程

```
systemctl disable vncserver@:display_number.service
systemctl stop vncserver@:display_number.service
```