Red Hat Enterprise Linux 9

使用 systemd 单元文件自定义和优化您的系统

使用 systemd 优化系统性能和扩展配置

Last Updated: 2024-06-26

Red Hat Enterprise Linux 9 使用 systemd 单元文件自定义和优化您的系统

使用 systemd 优化系统性能和扩展配置

法律通告

摘要

修改 systemd 单元文件并扩展默认配置，检查系统引导性能并优化 systemd 以缩短引导时间。

目录

对红帽文档提供反馈

我们感谢您对我们文档的反馈。让我们了解如何改进它。

通过 Jira 提交反馈（需要帐户）

1. 登录到 Jira 网站。
2. 在顶部导航栏中点 **Create**
3. 在 **Summary** 字段中输入描述性标题。
4. 在 **Description** 字段中输入您对改进的建议。包括到文档相关部分的链接。
5. 点对话框底部的 **Create**。

第 1 章 使用 SYSTEMD 单元文件

systemd 单元文件代表您的系统资源。作为系统管理员，您可以执行以下高级任务：

- 创建自定义单元文件
- 修改现有单元文件
- 使用实例化单元

1.1. 单元文件简介

单元文件包含描述这个单元并定义其行为的配置指令。几个 **systemctl** 命令在后台与单元文件一起工作。要进行更精细的调整，您可以手动编辑或创建单元文件。您可以在系统上找到存储单元文件的三个主要目录，**/etc/systemd/system/** 目录是为系统管理员创建或自定义的单元文件保留的。

单元文件名的格式如下：

> *<unit_name>.<type_extension>*

这里的 *unit_name* 代表单元名称，*type_extension* 标识单元类型。

例如，您可以找到系统上存在的 **sshd.service** 和 **sshd.socket** 单元。

可通过一个目录来补充单元文件，以了解额外的配置文件。例如，要将自定义配置选项添加到 **sshd.service** 中，请创建 **sshd.service.d/custom.conf** 文件，并在其中插入额外的指令：有关配置目录的更多信息，请参阅 修改现有单元文件。

systemd 系统和服务管理器也可以创建 **sshd.service.wants/** 和 **sshd.service.requires/** 目录。这些目录包含到 **sshd** 服务依赖的单元文件的符号链接。**systemd** 可以在安装过程中根据 [Install] 单元文件选项或在运行时根据 [Unit] 选项自动创建符号链接。您还可以手动创建这些目录和符号链接。

另外, **sshd.service.wants/** 和 **sshd.service.requires/** 目录可以被创建。这些目录包含到 **sshd** 服务依赖的单元文件的符号链接。符号链接会在安装过程中根据 [Install] 单元文件选项自动创建，或者根据 [Unit] 选项在运行时自动创建。也可以手动创建这些目录和符号链接。有关 [Install] 和 [Unit] 选项的详情请参考下表。

可以使用所谓的 **单元指定符** 来设置许多单元文件选项 - 在加载单元文件时，通配符字符串被动态地替换为单元参数。这可创建通用单元文件，来用作生成实例化单元的模板。请参阅使用实例化单元。

1.2. SYSTEMD 单元文件位置

您可以在以下目录中找到单元配置文件：

表 1.1. systemd 单元文件位置

目录	描述
/usr/lib/systemd/system/	与安装的 RPM 软件包一起分发的 **systemd** 单元文件。
/run/systemd/system/	在运行时创建的 **systemd** 单元文件。该目录优先于安装了的服务单元文件的目录。

目录	描述
/etc/systemd/system/	使用 **systemctl enable** 命令创建的 **systemd** 单元文件，以及为扩展服务添加的单元文件。这个目录优先于带有运行时单元文件的目录。

systemd 的默认配置在编译过程中定义，您可以在 **/etc/systemd/system.conf** 文件中找到配置。通过编辑此文件，您可以通过全局覆盖 **systemd** 单元的值来修改默认配置。

例如，若要覆盖设为 90 秒的超时限制的默认值，可使用 **DefaultTimeoutStartSec** 参数输入所需的值（以秒为单位）。

```
DefaultTimeoutStartSec=required value
```

1.3. 单元文件结构

单元文件通常由三个以下部分组成：

[Unit] 部分

包含不依赖于单元类型的通用选项。这些选项提供了单元描述，指定了单元的行为，并设置了其他单元的依赖项。有关最常用的 [Unit] 选项的列表，请参阅 重要 [单元] 部分选项。

[Unit type] 部分

包含特定于类型的指令，这些指令被分组在以单元类型命名的部分下。例如，服务单元文件包含 **[Service]** 部分。

[Install] 部分

包含有关 **systemctl enable** 和 **disable** 命令使用的单元安装的信息。有关 **[Install]** 部分的选项列表，请参阅 重要 [Install] 部分选项。

其他资源

- 重要 [Unit] 部分选项
- 重要 [Service] 部分选项
- 重要 [Install] 部分选项

1.4. 重要 [UNIT] 部分选项

下表列出了 [Unit] 部分的重要选项。

表 1.2. 重要 [Unit] 部分选项

选项 [a]	描述
描述	单元的有意义的描述。这个文本显示在 **systemctl status** 命令的输出中。
Documentation	提供单元参考文档的 URI 列表。

选项 [a]	描述
之后[b]	定义启动单位的顺序。这个单元仅在 **After** 中指定的单元处于活跃状态后才启动。与 **Requires** 不同，**After** 不会显式激活指定的单元。**Before** 选项与 **After** 的功能相反。
Requires	配置其它单元上的依赖关系。**Requires** 中列出的单元与单元一同被激活。如果任何需要的单元无法启动，则该单位就不会被激活。
期望	配置比 **Requires** 更弱的依赖项。如果列出的单元没有成功启动，它对单元激活不会有影响。这是建立自定义单元依赖项的建议方法。
Conflicts	配置负的依赖关系，与 **Requires** 相反。

[a] 有关 [Unit] 部分中可配置的选项的完整列表，请查看 **systemd.unit(5)** 手册页。

[b] 在大多数情况下，只需要**After** 和 **Before** 单元文件选项设置顺序依赖关系就足够了。如果还使用 **Wants**（推荐）或 **Requires**设置了需要的依赖关系，仍需要指定依赖关系顺序。这是因为排序和要求依赖关系可以独立地工作。

1.5. 重要 [SERVICE] 部分选项

下表列出了 [Service] 部分的重要选项。

表 1.3. 重要 [Service] 部分选项

选项 [a]	描述

选项 [a]	描述
Type	配置影响 **ExecStart** 功能的单元进程启动类型和相关选项。其中之一： * **simple** - 默认值。使用 **ExecStart** 启动的进程是该服务的主要进程。 * **forking** - 使用 **ExecStart** 启动的进程生成一个成为服务主进程的子进程，。父进程在启动完成后会退出。 * **oneshot** – 这个类型与 **simple** 类似，但在启动相应单位前会退出。 * **dbus** - 这个类型与 **simple** 类似，但后续单元仅在主进程获得 D-Bus 名称后启动。 * **notify** - 此类型与 **simple** 类似，但只有在通过 sd_notify（）函数发送通知消息后，后续单元才启动。 * **idle** - 与 **simple** 类似，服务二进制文件的实际执行会延迟，直到所有作业都完成，这避免了将状态输出与服务的 shell 输出混在一起。
ExecStart	指定在启动该单元时要执行的命令或脚本。**ExecStartPre** 和 **ExecStartPost** 指定在 **ExecStartPtart** 之前和之后要执行的自定义命令。**Type=oneshot** 启用指定可按顺序执行的多个自定义命令。
ExecStop	指定在该单元停止时要执行的命令或脚本。
ExecReload	指定重新载入该单元时要执行的命令或脚本。
Restart	启用此选项后，服务会在进程退出后重新启动，但 **systemctl** 命令的完全停止除外。
RemainAfterExit	如果设置为 True，即使所有进程都退出了，该服务也被视为活动状态。默认值为 False。这个选项在配置了 **Type=oneshot** 时特别有用。

[a] 有关 [Service] 部分中可配置的选项的完整列表，请参阅 **systemd.service(5)** 手册页。

1.6. 重要 [INSTALL] 部分选项

下表列出了 [Install] 部分的重要选项。

表 1.4. 重要 [Install] 部分选项

选项 [a]	描述
Alias	为这个单元提供空格分开的额外名称列表。除 **systemctl enable** 以外，多数**systemctl** 命令可使用别名而不是实际的单元名称。
RequiredBy	依赖于这个单元的单元列表。当启用此单元时，在 **RequiredBy** 中列出的单元会获得对这个单元的一个 **Require** 依赖项。
WantedBy	依赖于这个单元的单位列表。当启用这个单元时，在 **WantedBy** 中列出的单元会得到一个 **Want** 依赖项。
Also	指定要随这个单元一起安装或卸载的单元列表。
DefaultInstance	仅限于实例化单元，这个选项指定启用单位的默认实例。请参阅使用实例化单元。

[a] 有关 [Install] 部分中可配置的选项的完整列表，请查看 **systemd.unit(5)** 手册页。

1.7. 创建自定义单元文件

从头开始创建单元文件有多种用例：您可以运行自定义守护进程，创建某些现有服务的第二个实例，如使用 sshd 服务的第二个实例来创建自定义单元文件。

另一方面，如果您只想修改或扩展现有单元的行为，请使用 修改现有单元文件 中的说明。

步骤

1. 要创建一个自定义服务，请准备带有服务的可执行文件。文件可以包含自定义的脚本，也可以是软件供应商提供的可执行文件。如果需要，准备 PID 文件来保存自定义服务主要进程的恒定 PID。您还可以包含存储服务的 shell 变量的环境文件。确保源脚本是可执行的（通过执行 **chmod a+x**）且不是交互的。

2. 在 **/etc/systemd/system/** 目录中创建一个单元文件，并确定它有正确的文件权限。以 **root** 用户身份执行：

   ```
   # touch /etc/systemd/system/<name>.service

   # chmod 664 /etc/systemd/system/<name>.service
   ```

   将 *<name>* 替换为您要创建的服务的名称。请注意，文件不必是可执行的。

3. 打开创建的 ***<name>*.service** 文件，并添加服务配置选项。您可以根据您要创建的服务类型使用各种选项，请参阅 单元文件结构。
   以下是网络相关服务的单元配置示例：

   ```
   [Unit]
   Description=<service_description>
   ```

```
After=network.target

[Service]
ExecStart=<path_to_executable>
Type=forking
PIDFile=<path_to_pidfile>

[Install]
WantedBy=default.target
```

- *<service_description>* 是一个信息性描述，显示在 journal 日志文件和 **systemctl status** 命令的输出中。
- **After** 设置确保服务仅在网络运行后启动。添加一个空格分隔的其他相关服务或目标的列表。
- *path_to_executable* 代表到实际可执行服务的路径。
- **Type=forking** 用于进行 fork 系统调用的守护进程。该服务的主要进程使用 *path_to_pidfile* 中指定的 PID 创建。在 重要 [Service] 部分选项中查找其他启动类型。
- **WantedBy** 指出服务应在其下启动的目标。将这些目标视为旧的运行级别概念的替代品。

4. 通知 **systemd** 存在一个新的 ***<name>*.service** 文件：

```
# systemctl daemon-reload

# systemctl start <name>.service
```

警告

在创建新单元文件或修改现有单元文件后，始终执行 **systemctl daemon-reload** 命令。否则，**systemctl start** 或 **systemctl enable** 命令可能会因为 **systemd** 状态和磁盘上的实际的服务单元文件不匹配而失败。请注意，对于有大量单元的系统来说，这需要很长时间，因为每个单元的状态必须在重新载入的过程中被序列化，然后再进行反序列化。

1.8. 使用 SSHD 服务的第二个实例创建一个自定义单元文件

如果您需要配置和运行服务的多个实例，您可以创建原始服务配置文件的副本并修改某些参数，以避免与服务的主实例冲突。

步骤

要创建 **sshd** 服务的第二个实例：

1. 创建第二个守护进程将使用的 **sshd_config** 文件的一个副本：

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

2. 编辑上一步中创建的 **sshd-second_config** 文件，为第二个守护进程分配不同的端口号和 PID 文件：

```
Port 22220
PidFile /var/run/sshd-second.pid
```

有关 **Port** 和 **PidFile** 选项的更多信息，请参阅 **sshd_config**(5)手册页。请确定您选择的端口没有被其他服务使用。在运行该服务前，PID 文件不一定存在，它会在服务启动时自动生成。

3. 为 **sshd** 服务创建一个 **systemd** 单元文件副本：

```
# cp /usr/lib/systemd/system/sshd.service /etc/systemd/system/sshd-second.service
```

4. 更改创建的 **sshd-second.service** ：

   a. 修改 **Description** 选项：

   ```
   Description=OpenSSH server second instance daemon
   ```

   b. 将 **sshd.service** 添加到 **After** 选项中指定的服务，以便第二个实例仅在第一个实例启动后启动：

   ```
   After=syslog.target network.target auditd.service sshd.service
   ```

   c. 删除 **ExecStartPre=/usr/sbin/sshd-keygen** 行，**sshd** 的第一个实例包括密钥生成。

   d. 为 **sshd** 命令添加 **-f /etc/ssh/sshd-second_config** 参数，以便使用其它配置文件：

   ```
   ExecStart=/usr/sbin/sshd -D -f /etc/ssh/sshd-second_config $OPTIONS
   ```

   e. 修改后，**sshd-second.service** 单元文件包含以下设置：

   ```
   [Unit]
   Description=OpenSSH server second instance daemon
   After=syslog.target network.target auditd.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，请将第二个 **sshd** 实例的端口添加到 SSH 端口，否则拒绝将 **sshd** 的第二个实例绑定到端口：

```
# semanage port -a -t ssh_port_t -p tcp 22220
```

6. 启用 **sshd-second.service**，来在引导时自动启动：

```
# systemctl enable sshd-second.service
```

7. 使用 **systemctl status** 命令验证 **sshd-second.service** 是否正在运行。

8. 通过连接到该服务来验证是否正确启用了端口：

```
$ ssh -p 22220 user@server
```

确保配置了防火墙，以允许到 **sshd** 的第二个实例的连接。

1.9. 查找 SYSTEMD 服务描述

您可以在以 **#description** 开头的行中找到有关脚本的描述性信息。将此描述与单元文件的 [Unit] 部分中的 **Description** 选项中的服务名称一同使用。标头可能在 **#Short-Description** 和 **#Description** 行上包含类似的数据。

1.10. 查找 SYSTEMD 服务的依赖项

Linux 标准基础(LSB)标头可能包含多个在服务间构成依赖关系的指令。大多数可以转换为 systemd 单元选项，请查看下表：

表 1.5. LSB 标头中的依赖项选项

LSB 选项	描述	单元文件的对等
Provides	指定服务的引导工具名称，可在其他初始化脚本中引用（使用"$"前缀）。因为单元文件根据文件名指向其他单元，所以不再需要这个操作。	–
Required-Start	包含所需服务的引导工具名称。这被转换为一个排序依赖关系，引导工具名被替换为其所属的相应服务或目标的单元文件名。例如，如果是 **postfix**，$network 上的 Required-Start 依赖关系被转换为 network.target 上的 After 依赖关系。	**After**, **Before**
Should-Start	比 Required-Start 更弱的依赖项。Should-Start 依赖项失败不会影响服务的启动。	**After**, **Before**
required-Stop, **Should-Stop**	组成负依赖关系。	**Conflicts**

1.11. 查找服务的默认目标

以 **#chkconfig** 开始的行包含三个数字值。最重要的是第一个代表启动该服务的默认运行级别的数字。将这些运行级别映射到等同的 systemd 目标。然后在单元文件的 [Install] 部分中的 **WantedBy** 选项中列出这些目标。例如： **postfix** 之前在运行级别 2、3、4 和 5 中启动，它们转换为 multi-user.target 和 graphical.target。请注意，graphical.target 依赖于 multiuser.target，因此不需要指定它们。您也可以在 LSB 标头中的 **#Default-Start** 和 **#Default-Stop** 行中找到有关默认和禁止运行级别的信息。

#chkconfig 行里指定的其他两个值代表初始化脚本的启动和关闭优先级。如果 systemd 加载了初始化脚本，则这些值由 **systemd** 解释，但没有等效的单元文件。

1.12. 查找该服务使用的文件

初始化脚本需要从专用目录中载入功能库，并允许导入配置、环境和 PID 文件。环境变量在初始化脚本标头中以 **#config** 开头的行中指定，它转换为 **EnvironmentFile** 单元文件选项。**#pidfile** 初始化脚本行中指定的 PID 文件使用 **PIDFile** 选项导入到单元文件中。

未包含在初始化脚本标头中的关键信息是该服务可执行文件的路径，以及该服务可能需要的一些其他文件。在以前的 Red Hat Enterprise Linux 版本中，init 脚本使用 Bash case 语句定义默认操作的服务行为，如 **start**, **stop**, 或 **restart**，以及自定义的操作。以下来自 **postfix** init 脚本的摘录显示了在服务启动时要执行的代码块。

```
conf_check() {
    [ -x /usr/sbin/postfix ] || exit 5
    [ -d /etc/postfix ] || exit 6
    [ -d /var/spool/postfix ] || exit 5
}

make_aliasesdb() {
 if [ "$(/usr/sbin/postconf -h alias_database)" == "hash:/etc/aliases" ]
 then
  # /etc/aliases.db might be used by other MTA, make sure nothing
  # has touched it since our last newaliases call
  [ /etc/aliases -nt /etc/aliases.db ] ||
   [ "$ALIASESDB_STAMP" -nt /etc/aliases.db ] ||
   [ "$ALIASESDB_STAMP" -ot /etc/aliases.db ] || return
  /usr/bin/newaliases
  touch -r /etc/aliases.db "$ALIASESDB_STAMP"
 else
  /usr/bin/newaliases
 fi
}

start() {
 [ "$EUID" != "0" ] && exit 4
 # Check that networking is up.
 [ ${NETWORKING} = "no" ] && exit 1
 conf_check
 # Start daemons.
 echo -n $"Starting postfix: "
 make_aliasesdb >/dev/null 2>&1
 [ -x $CHROOT_UPDATE ] && $CHROOT_UPDATE
 /usr/sbin/postfix start 2>/dev/null 1>&2 && success || failure $"$prog start"
 RETVAL=$?
 [ $RETVAL -eq 0 ] && touch $lockfile
        echo
 return $RETVAL
}
```

初始化脚本的可扩展性允许指定两个自定义函数，**start()** 函数块调用的 **conf_check()** 和 **make_aliasesdb()**。然后,上面的代码中提到几个外部文件和目录：主服务可执行文件 **/usr/sbin/postfix**、**/etc/postfix/** 和 **/var/spool/postfix/** 配置目录，以及 **/usr/sbin/postconf/** 目录。

systemd 只支持预定义的操作，但可以执行带有 **ExecStart**、**ExecStartPre**、**ExecStartPost**、**ExecStop** 和 **ExecReload** 选项的自定义的可执行文件。在 service start 中执行 **/usr/sbin/postfix** 以及支持脚本。转换复杂的初始化脚本需要了解脚本中每个语句的用途。其中一些语句特定于操作系统版本，因此您不需要转换它们。另一方面，在新环境中可能需要在单元文件、以及服务可执行文件和支持文件中进行一些调整，。

1.13. 修改现有单元文件

如果要修改现有的单元文件，请进到 **/etc/systemd/system/** 目录。请注意，您不应该修改系统存储在 **/usr/lib/systemd/system/** 目录中的默认单元文件。

步骤

1. 根据所需更改的程度，选择以下方法之一：

   - 在 **/etc/systemd/system/*<unit>*.d/** 中为补充配置文件创建一个目录。我们推荐在大多数用例中使用这个方法。您可以使用额外的功能扩展默认配置，同时仍然指向原始单元文件。因此，软件包升级引入的默认单元的更改会被自动应用。如需更多信息，请参阅扩展默认单元配置。

   - 在 **/etc/systemd/system/ 目录中创建 /usr/lib/systemd/system/ 目录** 中原始单元文件的一个副本，并进行修改。这个副本会覆盖原始文件，因此不会应用软件包更新带来的更改。这个方法对无论软件包更新都应保留的重要单元更改都很有用。有关详细信息，请参阅覆盖默认单元配置。

2. 要返回单元的默认配置，请删除 **/etc/systemd/system/** 目录中自定义的配置文件。

3. 对单元文件应用更改，而不重启系统：

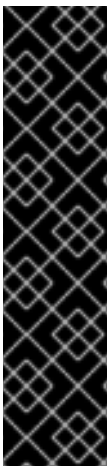
   ```
   # systemctl daemon-reload
   ```

   daemon-reload 选项重新加载所有单元文件，并重新创建依赖项树，这需要立即将任何更改应用到单元文件中。作为一种替代方案，您可以使用以下命令获得同样的效果：

   ```
   # init q
   ```

4. 如果修改后的单元文件属于一个正在运行的服务，请重启服务：

   ```
   # systemctl restart <name>.service
   ```



重要

要修改由 SysV initscript 处理的服务的属性，如依赖项或超时，请不要修改 initscript 本身。相反，为服务创建一个 **systemd** 置入配置文件，如 扩展默认的单元配置 和 覆盖默认的单元配置 中所述。

然后，像普通的 **systemd** 服务那样管理该服务。

例如：要扩展 **network** 服务的配置，不要修改 **/etc/rc.d/init.d/network** initscript 文件。反之，创建新目录 **/etc/systemd/system/network.service.d/** 和一个 **systemd** drop-in 文件 **/etc/systemd/system/network.service.d/*my_config*.conf**。然后将修改的值放到 drop-in 文件中。注： **systemd** 知道 **network** 服务为 **network.service**，这就是为什么创建的目录必须名为 **network.service.d**

1.14. 扩展默认单元配置

您可以使用额外的 systemd 配置选项扩展默认单元文件。

步骤

1. 在 **/etc/systemd/system/** 中创建一个配置目录：

```
# mkdir /etc/systemd/system/<name>.service.d/
```

将 *<name>* 替换为您要扩展的服务的名称。语法适用于所有单元类型。

2. 创建一个带有 **.conf** 后缀的配置文件：

```
# touch /etc/systemd/system/name.service.d/<config_name>.conf
```

将 *<config_name>* 替换为配置文件的名称。此文件遵循普通单元文件结构，且您必须在合适的部分中指定所有指令，请参阅 单元文件结构。

例如，要添加自定义依赖项，请使用以下内容创建配置文件：

```
[Unit]
Requires=<new_dependency>
After=<new_dependency>
```

<new_dependency> 代表要被标记为依赖项的单元。另一个例子是主进程退出后重新启动服务的配置文件，延迟 30 秒：

```
[Service]
Restart=always
RestartSec=30
```

创建仅聚焦一个任务的小配置文件。这些文件可轻松地移动或者链接到其他服务的配置目录。

3. 对单元应用更改：

```
# systemctl daemon-reload
# systemctl restart <name>.service
```

例 1.1. 扩展 httpd.service 配置

要修改 **httpd.service** 单元，以便在启动 Apache 服务时自动执行自定义 shell 脚本，请执行以下步骤。

1. 创建目录和自定义配置文件：

```
# mkdir /etc/systemd/system/httpd.service.d/
```

```
# touch /etc/systemd/system/httpd.service.d/custom_script.conf
```

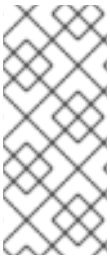
2. 通过将以下文本插入到 **custom_script.conf** 文件中，来指定您在主服务进程后要执行的脚本：

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

3. 应用单元更改：

```
# systemctl daemon-reload
```

```
# systemctl restart httpd.service
```



注意

/etc/systemd/system/ 配置目录中的文件优先于 **/usr/lib/systemd/system/** 中的单元文件。因此，如果配置文件包含一个只能指定一次的选项，如 **Description** 或 **ExecStart**，则此选项的默认值可被覆盖。请注意，在 **systemd-delta** 命令的输出中（在 Monitoring overrides units 中所述）中，比如这个单元总是被标记为 [EXTENDED]，即使总和和一些选项也会被覆盖。

1.15. 覆盖默认单元配置

您可以对更新提供单元文件的软件包后将保持不变的单元文件配置进行更改。

步骤

1. 以 **root** 用户身份输入以下命令来将单元文件复制到 **/etc/systemd/system/** 目录中：

```
# cp /usr/lib/systemd/system/<name>.service /etc/systemd/system/<name>.service
```

2. 使用文本编辑器打开复制的文件，并进行更改。

3. 应用单元更改：

```
# systemctl daemon-reload
# systemctl restart <name>.service
```

1.16. 更改超时限制

您可以为每个服务指定一个超时值，以防止出现故障的服务中断。否则，正常服务的超时默认值为 90 秒，SysV 兼容服务的超时默认值为 300 秒。

流程

要为 **httpd** 服务扩展超时限制：

1. 将 **httpd** 单元文件复制到 **/etc/systemd/system/** 目录中：

```
# cp /usr/lib/systemd/system/httpd.service /etc/systemd/system/httpd.service
```

2. 打开 **/etc/systemd/system/httpd.service** 文件，并在 **[Service]** 部分中指定 **TimeoutStartUSec** 值：

```
...
```

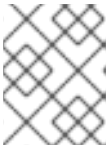
```
[Service]
...
PrivateTmp=true
TimeoutStartSec=10

[Install]
WantedBy=multi-user.target
...
```

3. 重新载入 **systemd** 守护进程：

```
# systemctl daemon-reload
```

4. Optional.验证新的超时值：

```
# systemctl show httpd -p TimeoutStartUSec
```



注意

要全局更改超时限制，在**/etc/systemd/system.conf** 中输入 **DefaultTimeoutStartSec**。

1.17. 监控覆盖的单元

您可以使用 **systemd-delta** 命令显示覆盖或修改的单元文件的概述。

流程

- 显示覆盖的或修改的单元文件的概述：

```
# systemd-delta
```

例如，命令的输出类似如下：

```
[EQUIVALENT] /etc/systemd/system/default.target → /usr/lib/systemd/system/default.target
[OVERRIDDEN] /etc/systemd/system/autofs.service →
/usr/lib/systemd/system/autofs.service

--- /usr/lib/systemd/system/autofs.service      2014-10-16 21:30:39.000000000 -0400
+++ /etc/systemd/system/autofs.service  2014-11-21 10:00:58.513568275 -0500
@@ -8,7 +8,8 @@
 EnvironmentFile=-/etc/sysconfig/autofs
 ExecStart=/usr/sbin/automount $OPTIONS --pid-file /run/autofs.pid
 ExecReload=/usr/bin/kill -HUP $MAINPID
-TimeoutSec=180
+TimeoutSec=240
+Restart=Always

 [Install]
 WantedBy=multi-user.target

[MASKED]    /etc/systemd/system/cups.service → /usr/lib/systemd/system/cups.service
[EXTENDED]   /usr/lib/systemd/system/sssd.service →
```

```
/etc/systemd/system/sssd.service.d/journal.conf

4 overridden configuration files found.
```

1.18. 使用实例化单元

您可以使用单个模板配置来管理一个服务的多个实例。您可以为单元定义一个通用模板，并在运行时使用特定参数生成该单元的多个实例。模板由 at 符号(@)表示。实例化的单元可以从另一个单元文件（使用 **Requires** 或者 **Wants** 选项）或者 **systemctl start 命令**启动。以下列方式命名实例化服务单元：

```
<template_name>@<instance_name>.service
```

<template_name> 代表模板配置文件的名称。将 *<instance_name>* 替换为单元实例的名称。多个实例可以指向带有通用于单元所有实例的配置选项的同一个模板文件。模板单元名称具有以下格式：

```
<unit_name>@.service
```

例如，单位文件中的以下 **Wants** 设置：

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

首先为给定服务单元进行 systemd 搜索。如果没有找到这样的单元，"@" 和类型后缀间的部分会被忽略，**systemd** 搜索 **getty@.service** 文件，从中读取配置并启动服务。

例如, **getty@.service** 模板包含以下指令：

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

当从上述模板实例化 **getty@ttyA.service** 和 **getty@ttyB.service** 时，**Description**= 被解析为 **Getty on ttyA** 和 **Getty on ttyB**。

1.19. 重要单元指定符

您可以在任何单元配置文件中使用通配符，称为 **单元指定符**。单元指定符替换某些单元参数，并在运行时被解释。

表 1.6. 重要单元指定符

单元指定符	含义	描述
%n	完整单元名称	代表完整的单元名称，包括类型后缀。**%N** 具有相同的含义，但也将禁用的字符替换为 ASCII 代码。
%p	前缀名称	代表删除了类型后缀的单元名称。对于实例化单元，%p 代表"@"字符前的单元名称的部分。

单元指定符	含义	描述
%i	实例名称	是"@"字符和类型后缀之间的实例化单元名称的一部分。**%I** 具有相同的含义，但也会将禁用的字符替换为 ASCII 码。
%H	主机名	代表在载入单元配置时的运行系统的主机名。
%t	运行时目录	代表运行时目录，对于 **root** 用户是 **/run**，对于非特权用户是 XDG_RUNTIME_DIR 变量的值。

有关单元指定符的完整列表，请参见 **systemd.unit(5)** 手册页。

1.20. 其他资源

- 如何为 RHEL 和 systemd 中的服务设置限制
- 如何编写强制必须启动特定服务的服务单元文件
- 如何决定 systemd 服务单元定义应具有哪些依赖项

第 2 章 优化 SYSTEMD 以缩短引导时间

作为系统管理员，您可以优化系统的性能，并缩短引导时间。您可以查看启动过程中 **systemd** 启动的服务，并评估它们的必要性。禁用某些服务在引导时启动可以提高系统的引导时间。

2.1. 检查系统引导性能

要检查系统引导性能，您可以使用 **systemd-analyze 命令**。通过使用某些选项，您可以调优 systemd 来缩短引导时间。

先决条件

- 可选：在检查 **systemd** 以调整引导时间前，列出所有启用的服务：

```
$ systemctl list-unit-files --state=enabled
```

流程

选择您要分析的信息：

- 分析最后一次成功引导所花费的时间的信息：

```
$ systemd-analyze
```

- 分析每个 **systemd** 单元的单元初始化时间：

```
$ systemd-analyze blame
```

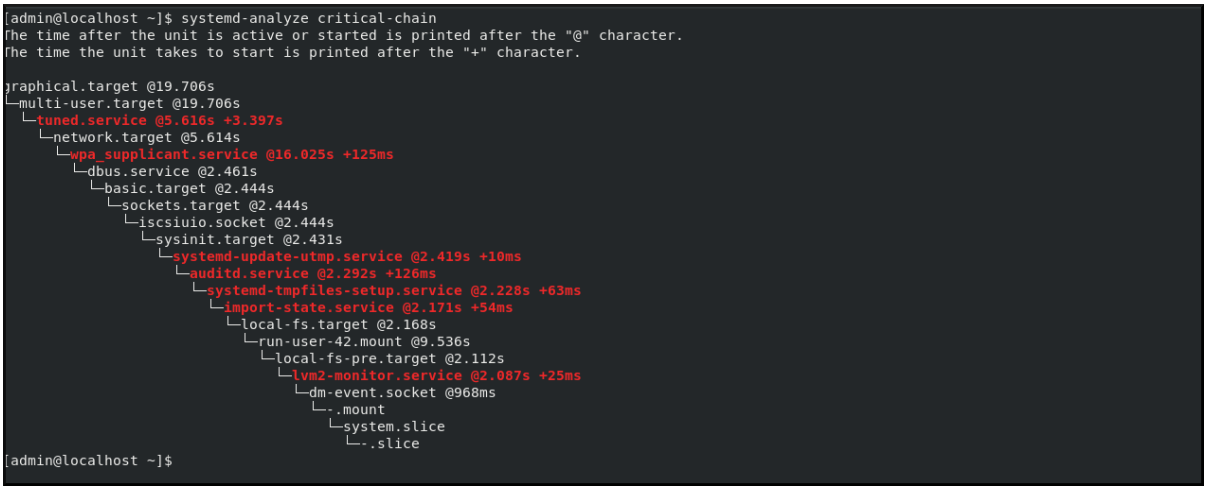
输出会根据在上一次成功引导过程中初始化的时间以降序列出。

- 识别在最后一次成功引导时花费最长时间初始化的关键单元：

```
$ systemd-analyze critical-chain
```

输出突出显示使用红色的引导速度非常慢的单元。

图 2.1. systemd-analyze critical-chain 命令的输出



其他资源

- **systemd-analyze (1)** 手册页

2.2. 为选择可安全禁用的服务提供指导信息

您可以通过禁用默认引导时启用的某些服务来缩短系统的引导时间。

- 列出启用的服务：

```
$ systemctl list-unit-files --state=enabled
```

- 禁用一个服务：

```
# systemctl disable <service_name>
```

某些服务必须保持启用，以便您的操作系统安全，且以您需要的方式正常工作。

请参阅下表，作为您选择可以安全禁用的服务的指南。此表列出了在 Red Hat Enterprise Linux 最小安装上默认启用的所有服务。

表 2.1. 在 RHEL 的最小安装中默认启用的服务

服务名称	它可用被禁用吗？	更多信息
auditd.service	是	仅在不需要内核提供审核信息时禁用 **auditd.service**。请注意，如果禁用 **auditd.service**，则不会生成 **/var/log/audit/audit.log** 文件。因此,您无法追溯检查一些常见的动作或事件，如用户登录、服务启动或密码更改。还请注意 auditd 有两个部分：内核部分和服务本身。使用 **systemctl disable auditd** 命令，您只是禁用了该服务，而不是禁用内核的部分。要禁用系统审核，请在内核命令行中设置 **audit=0**。
autovt@.service	否	这个服务只在真正需要时才运行，因此不需要禁用它。
crond.service	是	请注意，如果您禁用 crond.service，则不会运行 crontab 中的项目。
dbus-org.fedoraproject.FirewallD1.service	是	到 **firewalld.service** 的符号链接
dbus-org.freedesktop.NetworkManager.service	是	到 **NetworkManager.service** 的符号链接
dbus-org.freedesktop.nm-dispatcher.service	是	到 **NetworkManager-dispatcher.service**的符号链接
firewalld.service	是	仅在不需要防火墙时禁用 **firewalld.service**。

服务名称	它可用被禁用吗？	更多信息
getty@.service	否	这个服务只在真正需要时才运行，因此不需要禁用它。
import-state.service	是	仅在不需要从网络存储引导时才禁用 **import-state.service**。
irqbalance.service	是	仅在只有一个 CPU 时禁用 **irqbalance.service**。不要在有多个 CPU 的系统中禁用 **irqbalance.service**。
kdump.service	是	仅在不需要内核崩溃报告时禁用 **kdump.service**。
loadmodules.service	是	除非 **/etc/rc.modules** 或 **/etc/sysconfig/modules** 目录存在，否则不会启动该服务，这意味着它不会在最小 RHEL 安装中启动。
lvm2-monitor.service	是	仅在不使用逻辑卷管理器(LVM)时禁用 **lvm2-monitor.service**。
microcode.service	否	不要禁用该服务，因为它在 CPU 中提供了 microcode 软件的更新。
NetworkManager-dispatcher.service	是	只在不需要在网络配置更改时通知时才禁用 **NetworkManager-dispatcher.service**（例如在静态网络中）。
NetworkManager-wait-online.service	是	只有在引导后不需要工作网络连接时才禁用 **NetworkManager-wait-online.service**。如果启用该服务，则该系统不会在网络连接正常工作前完成引导。这可能会大大延长引导时间。
NetworkManager.service	是	仅在不需要连接到网络时禁用 **NetworkManager.service**。
nis-domainname.service	是	仅在不使用网络信息服务（NIS）时禁用 **nis-domainname.service**。
rhsmcertd.service	否	
rngd.service	是	仅在系统上不需要大量熵或者没有任何硬件生成器时禁用 **rngd.service**。请注意，在需要大量好熵的环境中，比如用于生成 X.509 证书的系统（如 FreeIPA 服务器）中，该服务是必需的。
rsyslog.service	是	仅在不需要持久性日志，或把 **systemd-journald** 设置为持久性模式时，禁用 **rsyslog.service**。
selinux-autorelabel-mark.service	是	仅在不使用 SELinux 时禁用 **selinux-autorelabel-mark.service**。
sshd.service	是	仅在不需要 OpenSSH 服务器远程登录时禁用 **sshd.service**。
sssd.service	是	仅在没有通过网络登录系统的用户（例如，使用 LDAP 或 Kerberos）时禁用 **sssd.service**。如果禁用了 **sssd.service**，红帽建议禁用所有 **sssd-*** 单元。

服务名称	它可用被禁用吗？	更多信息
syslog.service	是	**rsyslog.service** 的别名
tuned.service	是	仅在需要使用性能调整时禁用 **tuned.service**。
lvm2-lvmpolld.socket	是	仅在您不使用逻辑卷管理器（LVM）时禁用 **lvm2-lvmpolld.socket**。
dnf-makecache.timer	是	仅在不需要自动更新软件包元数据时禁用 **dnf-makecache.timer**。
unbound-anchor.timer	是	仅在不需要每日更新 DNS 安全扩展（DNSSEC）的根信任锚时禁用 **unbound-anchor.timer**。Unbound resolver 和 resolver 库使用这个根信任锚器进行 DNSSEC 验证。

要查找服务的更多信息，请使用以下命令之一：

```
$ systemctl cat <service_name>
```

```
$ systemctl help <service_name>
```

systemctl cat 命令提供相应 **/usr/lib/systemd/system/*<service>*** 服务文件的内容，以及所有适用的覆盖。适用的覆盖包括 **/etc/systemd/system/*<service>*** 文件中的单元文件覆盖，或者相应的 **unit.type.d** 目录中的流动文件。

其他资源

- **systemd.unit (5)** 手册页
- 显示特定服务手册页的 **systemd help** 命令

2.3. 其他资源

- **systemctl**(1)手册页
- **systemd**(1)手册页
- **systemd-delta**(1)手册页
- **systemd.directives**(7)手册页
- **systemd.unit**(5)手册页
- **systemd.service**(5)手册页
- **systemd.target**(5)手册页
- **systemd.kill**(5)手册页

- systemd 主页