

# 18. Customized service management

## 18.1 Create a new service file

Each service has its own corresponding configuration file in Linux. This file can be edited with a text editor and has an extension of `xxx.service` (xxx is the service name). These files are located in the `/usr/lib/systemd/system` directory.

Create our service by creating a new service file in this directory. The content of the file is structured as follows:

```
[Unit]
Description=service description
After = service dependencies (start this service after these services)

[Service]
Type=service type
ExecStart=Start command
ExecStop=Stop command
ExecReload=Restart command

[Install]
WantedBy=Service installation settings
```

It can be seen that the service configuration file is divided into three parts: `[Unit]`, `[Service]` and `[Install]`.

Generally speaking, some values are fixed, and we can apply them directly if there are no special needs. For example, the value of `After` in `[Unit]` is generally: `network.target remote-fs.target nss-lookup.target`.

`WantedBy` of `[Install]` is usually `multi-user.target`.

`[Service]` is the main content.

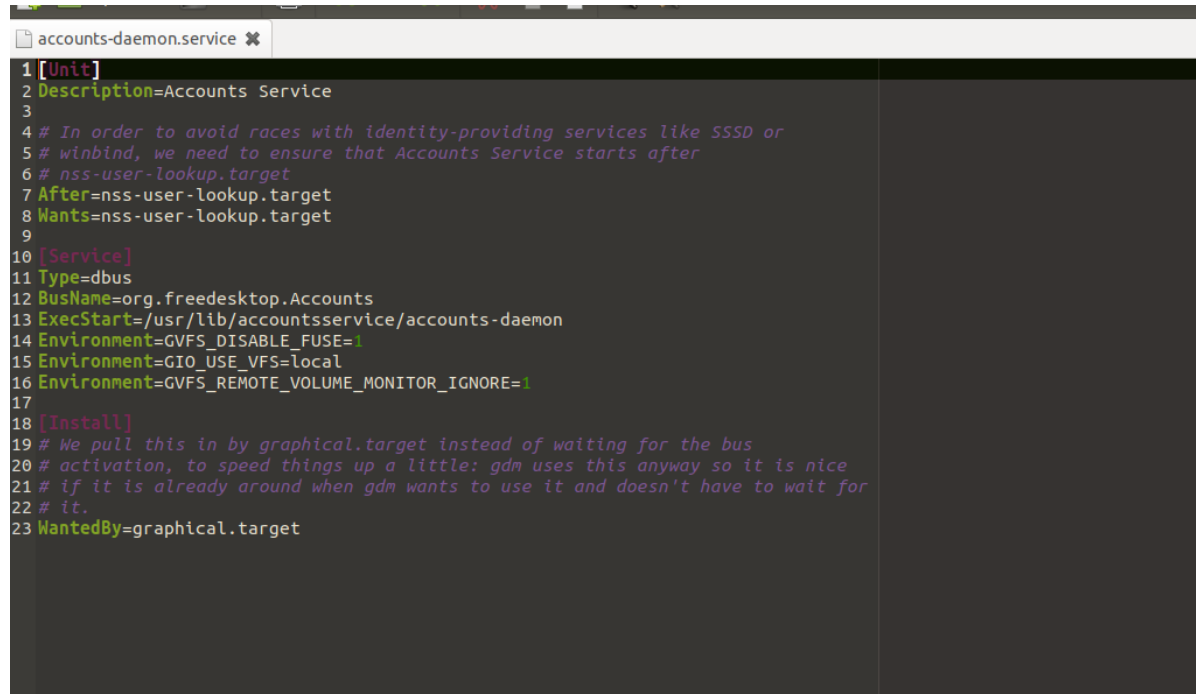
The values of `Type` are as follows:

- `simple`: This is the default value. After the `ExecStart` setting is specified, `simple` is the default `Type` setting unless `Type` is specified. `simple` uses the process created by `ExecStart` as the main process of the service. Under this setting, `systemd` will start the service immediately.
- `forking`: If this value is used, the `ExecStart` script will call the `fork()` function to create a process as part of its startup. When initialization is complete, the parent process will exit. The child process will continue to execute as the main process.
- `oneshot`: Similar to `simple`, but the process will exit before `systemd` is started. This is a one-time act. You may also need to set `RemainAfterExit=yes` so that `systemd` thinks that the process is still active after exiting.
- `dbus`: also very similar to `simple`, this configuration expects or sets a name value, just set the name by setting `BusName=`.
- `notify`: Again, similar configuration to `simple`. As the name suggests, this setting will send push messages when the daemon starts.

In fact, the commonly used ones are `simple` and `forking`. Generally speaking, if our program is used in the frontend of the application, we use `simple`, and the background/daemon process is usually `forking`.

Then there are the start/stop/restart commands. Note that all programs called in this command must use absolute paths.

For example, the redis Service configuration on my server:

A screenshot of a terminal window with a dark background. The title bar shows a file named 'accounts-daemon.service'. The terminal displays the contents of this file, which is a systemd service unit. The configuration includes a [Unit] section with a description and dependencies, a [Service] section with type, bus name, and exec start settings, and an [Install] section with WantedBy. The text is color-coded: [Unit] is red, [Service] is green, and [Install] is blue. Comments are in purple.

```
1 [Unit]
2 Description=Accounts Service
3
4 # In order to avoid races with identity-providing services like SSSD or
5 # winbind, we need to ensure that Accounts Service starts after
6 # nss-user-lookup.target
7 After=nss-user-lookup.target
8 Wants=nss-user-lookup.target
9
10 [Service]
11 Type=dbus
12 BusName=org.freedesktop.Accounts
13 ExecStart=/usr/lib/accounts-service/accounts-daemon
14 Environment=GVFS_DISABLE_FUSE=1
15 Environment=GIO_USE_VFS=local
16 Environment=GVFS_REMOTE_VOLUME_MONITOR_IGNORE=1
17
18 [Install]
19 # We pull this in by graphical.target instead of waiting for the bus
20 # activation, to speed things up a little: gdm uses this anyway so it is nice
21 # if it is already around when gdm wants to use it and doesn't have to wait for
22 # it.
23 WantedBy=graphical.target
```

This is just an example and does not fill in the end command and repeat command.

## 18.2 Start/stop/restart our service

We have just set up our service configuration and it is ready to use! Before doing this, you need to use the following command to let the system re-read all service files:

```
systemctl daemon-reload
```

Then, control the service with the following command:

```
# Start service
service service name start

# Terminate service
service service name stop

# Restart the service
service service name restart
```

Note that the service name is the file name of the service configuration file service file we just created (excluding the extension). For example, my service file is `redis-server.service`, then my service name is `redis-server`.

In fact, when we execute the start service command, the command with the value of `ExecStart` in the configuration file we just executed will be executed. Similarly, the termination and restart will correspond to the commands with the value of `ExecStop` and `ExecReload` in the configuration file.

## 18.3 Enable/disable auto-start at boot

Enable/disable auto-start at boot with the following command:

```
# Enable auto-start at boot
systemctl enable service name

# Disable auto-start at boot
systemctl disable service name
```