# Red Hat Enterprise Linux 9

## 安装和使用动态编程语言

在 Red Hat Enterprise Linux 9 中安装和使用动态编程语言的指南

Last Updated: 2022-05-19

# Red Hat Enterprise Linux 9 安装和使用动态编程语言

在 Red Hat Enterprise Linux 9 中安装和使用动态编程语言的指南

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

## 法律通告



## 摘要

本文档论述了安装和使用动态编程语言的基础知识，如 Red Hat Enterprise Linux 9 上的 Python 和 PHP。

# 目录

# 让开源更具包容性

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

# 对红帽文档提供反馈

我们感谢您对文档提供反馈信息。请让我们了解如何改进文档。

- 关于特定内容的简单评论：
  1. 请确定您使用 *Multi-page HTML* 格式查看文档。另外，确定 **Feedback** 按钮出现在文档页的右上方。
  2. 用鼠标指针高亮显示您想评论的文本部分。
  3. 点在高亮文本上弹出的 **Add Feedback**。
  4. 按照显示的步骤操作。
- 要通过 Bugzilla 提交反馈，请创建一个新的 ticket：
  1. 进入 Bugzilla 网站。
  2. 在 Component 中选择 **Documentation**。
  3. 在 **Description** 中输入您要提供的信息。包括文档相关部分的链接。
  4. 点 **Submit Bug**。

# 第 1 章 PYTHON 简介

Python 是一种高级编程语言，支持多种编程模式，如面向对象、所需功能以及程序式范式。Python 具有动态语义，可用于通用编程。

使用 Red Hat Enterprise Linux 时，系统中安装的许多软件包（如提供系统工具、数据分析工具或 Web 应用程序的软件包）会使用 Python 编写。要使用这些软件包，您必须安装 **python*** 软件包。

## 1.1. PYTHON 版本

**Python 3.9** 是 RHEL 9 中的默认 **Python** 实现。**Python 3.9** 在 BaseOS 存储库中的非模块化 **python3** RPM 软件包中分发，通常默认安装。**Python 3.9** 将支持 RHEL 9 的整个生命周期。

未来，其它版本的 **Python 3** 将作为 RPM 软件包发布，且带有较短的生命周期（通过 AppStream 软件仓库）。这些版本将与 Python 3.9 并行安装。

**Python 2** 不随 RHEL 9 提供。

## 1.2. 自 RHEL 8 开始的 PYTHON 生态系统的主要区别

本节总结了 RHEL 9 中 Python 生态系统与 RHEL 8 相比的显著变化。

### unversioned python 命令

**python** 命令的未指定版本形式(**/usr/bin/python**)在 **python-unversioned-command** 软件包中提供。在某些系统中，默认情况下不安装此软件包。要手动安装 **python** 命令的未指定版本形式，请使用 **dnf install /usr/bin/python** 命令。

在 RHEL 9 中，**python** 命令的未指定版本形式指向默认的 **Python 3.9** 版本，它相当于 **python3** 和 **python3.9** 命令。

**python** 命令用于交互式会话。在生产环境中，红帽建议明确使用 **python3** 或 **python3.9**。

您可以使用 **dnf remove /usr/bin/python** 命令卸载未指定版本的 **python** 命令。

如果需要不同的 python 命令，您可以在 **/usr/local/bin** 或 **~/.local/bin** 中创建自定义符号链接。

还有其他未指定版本的命令，如 **python3- pip 软件包中的 /usr/bin/ pip**。在 RHEL 9 中，所有未指定版本的命令都指向默认的 **Python 3.9** 版本。

### 特定于架构的 Python wheels

在 RHEL 9 上构建的特定于体系结构的 Python **wheel** 新建了上游架构命名，允许客户在 RHEL 9 上构建其 Python **wheel** 并在非 RHEL 系统中安装它们。在以前的 RHEL 版本构建的 Python **wheel** 是向前兼容的，可以在 RHEL 9 上安装。请注意，这仅影响包含 Python 扩展的 **wheel**，这些扩展针对每个架构构建，而不影响包含纯 Python 代码的 Python **wheels**，这不是特定于架构的 Python wheel。

# 第 2 章 安装和使用 PYTHON

在 RHEL 9 中，**Python 3.9** 是默认的 **Python** 实施。unversioned **python** 命令指向默认的 **Python 3.9** 版本。

## 2.1. 安装 PYTHON 3

默认 Python 实现通常会默认安装。要手动安装它，请使用以下步骤。

**步骤**

- 要安装 Python，请使用：

```
# dnf install python3
```

**验证步骤**

- 要验证系统中安装的 Python 版本，请使用以下命令：

```
$ python3 --version
```

## 2.2. 安装其他 PYTHON 3 软件包

前缀为 **python3** 的软件包包含默认 **Python 3.9** 版本的模块。

**步骤**

- 要为 Python 安装 **Requests** 模块，请使用：

```
# dnf install python3-requests
```

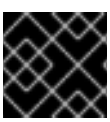
- 要从 Python 安装 **pip** 软件包安装程序，请使用：

```
# dnf install python3-pip
```

## 2.3. 为开发人员安装其他 PYTHON 3 工具

其他面向开发人员的 Python 工具通过 CodeReady Linux Builder 存储库发布。

此存储库包含 **python3-pytest**、**python3-Cython** 软件包等内容。



**重要**

红帽不支持 CodeReady Linux Builder 存储库及其内容。

要从存储库中安装软件包，请使用以下步骤。

**步骤**

1. 启用 CodeReady Linux Builder 存储库：

```
# subscription-manager repos --enable codeready-builder-for-rhel-9-x86_64-rpms
```

2. 安装 **python3-pytest** 软件包：

```
# dnf install python3-pytest
```

**其他资源**

- 如何在 CodeReady Linux Builder 中启用和使用内容

## 2.4. 使用 PYTHON

以下流程包含运行 Python 解释器或 Python 相关命令的示例。

**先决条件**

- 确保已安装 Python。

**步骤**

- 要运行 Python 解释器或相关命令，请使用：

```
$ python3
$ python3 -m pip --help
$ python3 -m pip install package
```

# 第 3 章 打包 PYTHON 3 RPM

您可以使用 **pip** 安装程序，或使用 DNF 软件包管理器在系统中安装 Python 软件包。DNF 使用 RPM 软件包格式，它提供对软件的下游控制。

原生 Python 软件包的打包格式由 Python 打包授权机构(PyPA)规范定义。大多数 Python 项目使用 **distutils** 或 **setuptools** 实用程序进行打包，并在 **setup.py** 文件中定义的软件包信息。然而，创建原生 Python 软件包的可能性随着时间推移而演进。有关新兴打包标准的更多信息，请参阅 pyproject-rpm-macros。

本章论述了如何将 **setup.py** 的 Python 项目打包到一个 RPM 软件包中。与原生 Python 软件包相比，此方法提供以下优点：

- 可以对 Python 和非 Python 软件包的依赖项，并严格由 **DNF** 软件包管理器强制执行。
- 您可以用加密的方式为软件包签名。使用加密签名，您可以验证、集成和测试 RPM 软件包的内容与操作系统的其余部分。
- 您可以在构建过程中执行测试。

## 3.1. PYTHON 软件包的 SPEC 文件描述

SPEC 文件包含 **rpmbuild** 实用程序用于构建 RPM 的指令。这些说明包含在一系列部分中。SPEC 文件有两个主要部分，它们定义了该部分：

- preamble（包含一系列在 Body 中使用的元数据项）
- 正文（包含指令的主要部分）

与非 Python RPM SPEC 文件相比，适用于 Python 项目的 RPM SPEC 文件有一些特定信息。

**重要**

Python 库的任何 RPM 软件包的名称必须始终包含 **python3-** 前缀。

其他具体信息可在以下 **适用于 python3-pello 软件包 的 SPEC 文件示例**中显示。有关此类特定描述，请查看示例中的备注。

```
Name:           python-pello                                    1
Version:        1.0.2
Release:        1%{?dist}
Summary:        Example Python library

License:        MIT
URL:            https://github.com/fedora-python/Pello
Source:         %{url}/archive/v%{version}/Pello-%{version}.tar.gz

BuildArch:      noarch
BuildRequires:  python3-devel                                   2

# Build dependencies needed to be specified manually
BuildRequires:  python3-setuptools

# Test dependencies needed to be specified manually
```

```
# Also runtime dependencies need to be BuildRequired manually to run tests during build
BuildRequires:  python3-pytest >= 3


%global _description %{expand:
Pello is an example package with an executable that prints Hello World! on the command line.}

%description %_description

%package -n python3-pello                                    3
Summary:        %{summary}

%description -n python3-pello %_description


%prep
%autosetup -p1 -n Pello-%{version}


%build
# The macro only supported projects with setup.py
%py3_build                                                   4


%install
# The macro only supported projects with setup.py
%py3_install


%check                                                       5
%{pytest}


# Note that there is no %%files section for the unversioned python module
%files -n python3-pello
%doc README.md
%license LICENSE.txt
%{_bindir}/pello_greeting

# The library files needed to be listed manually
%{python3_sitelib}/pello/

# The metadata files needed to be listed manually
%{python3_sitelib}/Pello-*.egg-info/
```

1. 将 Python 项目打包到 RPM 中时，始终将 **python-** 前缀添加到项目的原始名称。这里的原始名称为 **pello**，因此 **源 RPM(SRPM)** 的名称是 **python-pello**。

2. **BuildRequires** 指定构建并测试此软件包所需的软件包。在 **BuildRequires** 中，始终包括提供构建 Python 软件包所需工具的项目： **python3-devel** 和您软件包所需的相关项目，如 **python3-setuptools** 或在 **%check** 部分中运行测试所需的运行时和测试依赖关系。

3. 当为二进制 RPM 选择名称时（用户可以安装的软件包）时，添加版本化的 Python 前缀，即当前 **python3-**。因此，生成的二进制 RPM 将命名为 **python3-pello**。

4.

**%py3_build** 和 **%py3_install** macros 宏分别运行 **setup.py build** 和 **setup.py install** 命令，使用附加参数来指定安装位置、要使用的解释器以及其他详情。

5. **%check** 部分应该运行打包项目的测试。确切的命令取决于项目本身，但可以使用 **%pytest** 宏以 RPM 友好的方式运行 **pytest** 命令。**%{python3}** 宏包含 Python 3 解释器的路径，即 **/usr/bin/python3**。我们建议使用宏，而不是字面上的路径。

## 3.2. PYTHON 3 RPM 的常见宏

在 SPEC 文件中，始终使用以下 Macros *用于 Python 3 RPM* 表而不是硬编码其值的宏。

**表 3.1. Python 3 RPM 宏**

| Macro | 常规定义 | 描述 |
|---|---|---|
| %{python3} | /usr/bin/python3 | Python 3 解释器 |
| %{python3_version} | 3.9 | Python 3 解释器的 major.minor 版本 |
| %{python3_sitelib} | /usr/lib/python3.9/site-packages | 安装纯 Python 模块的位置 |
| %{python3_sitearch} | /usr/lib64/python3.9/site-packages | 安装包含特定于架构扩展模块的模块的位置 |
| %py3_build | | 使用适用于 RPM 软件包的参数运行 **setup.py build** 命令 |
| %py3_install | | 使用适用于 RPM 软件包的参数运行 **setup.py install** 命令 |
| %{py3_shebang_flags} | s | Python 解释器指令宏的默认标记集，**%py3_shebang_fix** |
| %py3_shebang_fix | | 将 Python 解释器指令改为 **#! %{python3}**，保留任何现有标志（如果找到），并添加在 **%{py3_shebang_flags}** 宏中定义的标记 |

**其他资源**

- 上游文档中的 Python 宏

## 3.3. 为 PYTHON RPM 使用自动生成的依赖项

以下流程描述了如何在将 Python 项目打包为 RPM 时使用自动生成的依赖项。

**先决条件**

- RPM 的 SPEC 文件存在。如需更多信息，请参阅 Python 软件包的 SPEC 文件描述。

**步骤**

1. 确保以下包含上游提供元数据的目录之一包含在生成的 RPM 中：

   - **.dist-info**
   - **.egg-info**
     RPM 构建过程会自动从这些目录中生成虚拟 **pythonX.Ydist**，例如：

     ```
     python3.9dist(pello)
     ```

     然后，Python 依赖项生成器读取上游元数据，并使用生成的 **pythonX.Ydist** 虚拟提供为每个 RPM 软件包生成运行时要求。例如，生成的要求标签可能如下所示：

     ```
     Requires: python3.9dist(requests)
     ```

2. 检查生成的要求。
3. 要删除其中的一些生成的需要，请使用以下方法之一：

   a. 在 SPEC 文件的 **%prep** 部分中修改上游提供的元数据。

   b. 使用 上游文档 中描述的依赖项自动过滤。

4. 要禁用自动依赖项生成器，请在主软件包的 **%description** 声明中包含 **%{?python_disable_dependency_generator}** 宏。

**其他资源**

- 自动生成的依赖项

# 第 4 章 在 PYTHON 脚本中处理解释器指令

在 Red Hat Enterprise Linux 9 中，可执行 Python 脚本应该使用解析程序指令（也称为 hashbangs 或 shebangs），至少指定主要 Python 版本。例如：

```
#!/usr/bin/python3
#!/usr/bin/python3.9
```

在构建任何 RPM 软件包时，**/usr/lib/rpm/redhat/brp-mangle-shebangs** buildroot 策略(BRP)脚本会自动运行，并尝试在所有可执行文件中更正解释器指令。

当遇到带有模糊的解释解释器指令的 Python 脚本时，BRP 脚本会生成错误，例如：

```
#!/usr/bin/python
```

或者

```
#!/usr/bin/env python
```

## 4.1. 修改 PYTHON 脚本中的解释器指令

使用以下步骤修改 Python 脚本中的解释器指令，以便在 RPM 构建时出现错误。

**先决条件**

- Python 脚本中的一些解释器指令会导致构建错误。

**步骤**

- 要修改解释器指令，请完成以下任务之一：
  - 在您的 SPEC 文件的 **%prep** 部分中使用以下宏：

    ```
    # %py3_shebang_fix SCRIPTNAME …
    ```

    *SCRIPTNAME* 可以是任何文件、目录或文件和目录列表。

    因此，列出的所有文件以及列出目录中所有 **.py** 文件都会修改其解释器指令以指向 **%{python3}**。将保留原始解释器指令的现有标记，并将添加 **%{py3_shebang_flags}** 宏中定义的其他标志。您可以在 SPEC 文件中重新定义 **%{py3_shebang_flags}** 宏，以更改将要添加的标志。

  - 从 **python3-devel** 软件包应用 **pathfix.py** 脚本：

    ```
    # pathfix.py -pn -i %{python3} PATH …
    ```

    您可以指定多个路径。如果 ***PATH*** 是一个目录，则 **pathfix.py** 会递归扫描与模式 **^[a-zA-Z0-9_]+\.py$** 匹配的 Python 脚本，而不仅仅是具有模糊的解释器指令。将上述命令添加到 **%prep** 部分，或者在 **%install** 部分的末尾。

  - 修改打包的 Python 脚本，以便它们符合预期格式。为此，您也可以使用 RPM 构建进程之外的 **pathfix.py** 脚本。在 RPM 构建之外运行 **pathfix.py** 时，将上面的示例中的 **%{python3}** 替换为解释器指令的路径，如 **/usr/bin/python3**。

**其它资源**

- 解释器调用

# 第 5 章 使用 PHP 脚本语言

超文本 Preprocessor(PHP)是主要用于服务器端脚本的通用脚本语言，可让您使用 Web 服务器运行 PHP 代码。

## 5.1. 安装 PHP 脚本语言

这部分论述了如何安装 PHP。

**步骤**

- 要安装 PHP，请使用：

```
# dnf install php
```

## 5.2. 通过 WEB 服务器使用 PHP 脚本语言

### 5.2.1. 在 Apache HTTP 服务器中使用 PHP

在 Red Hat Enterprise Linux 9 中，**Apache HTTP 服务器** 可让您将 PHP 作为 FastCGI 进程服务器运行。FastCGI Process Manager(FPM)是一种替代 PHP FastCGI 守护进程，它允许网站管理高负载。默认情况下，PHP 在 RHEL 9 中使用 FastCGI Process Manager。

本节论述了如何使用 FastCGI 进程服务器运行 PHP 代码。

**先决条件**

- 在您的系统上安装 PHP 脚本语言。

**步骤**

1. 安装 **httpd** 软件包：

```
# dnf install httpd
```

2. 启动 **Apache HTTP 服务器** ：

```
# systemctl start httpd
```

   或者，如果 **Apache HTTP** 服务器已在您的系统中运行，请在安装 PHP 后重启 **httpd** 服务：

```
# systemctl restart httpd
```

3. 启动 **php-fpm** 服务：

```
# systemctl start php-fpm
```

4. 可选：在引导时启用这两个服务：

```
# systemctl enable php-fpm httpd
```

5. 要获取有关 PHP 设置的信息，请在 **/var/www/html/** 目录中创建带有以下内容的 **index.php** 文件：

```
echo '<?php phpinfo(); ?>' > /var/www/html/index.php
```

6. 要运行 **index.php** 文件，请将浏览器指向：

```
http://<hostname>/
```

7. 可选：如果您有特定要求，请调整配置：

- **/etc/httpd/conf/httpd.conf** - 一般的 **httpd** 配置
- **/etc/httpd/conf.d/php.conf** - **httpd**特定 PHP 配置
- **/usr/lib/systemd/system/httpd.service.d/ php-fpm.conf** - 默认情况下，php-fpm 服务使用 **httpd**启动
- **/etc/php-fpm.conf** - FPM 主配置
- **/etc/php-fpm.d/www.conf** - 默认 **www** 池配置

**例 5.1. 运行"Hello, World!" 使用 Apache HTTP 服务器的 PHP 脚本**

1. 在 **/var/www/html/** 目录中为您的项目创建一个 **hello** 目录：

```
# mkdir hello
```

2. 在 **/var/www/html/hello/** 目录中创建 **hello.php** 文件，其内容如下：

```
# <!DOCTYPE html>
<html>
<head>
<title>Hello, World! Page</title>
</head>
<body>
<?php
    echo 'Hello, World!';
?>
</body>
</html>
```

3. 启动 **Apache HTTP 服务器** ：

```
# systemctl start httpd
```

4. 要运行 **hello.php** 文件，请将浏览器指向：

```
http://<hostname>/hello/hello.php
```

因此，会显示带有 "Hello, World!" 文本的网页。

**其它资源**

- 设置 Apache HTTP web 服务器

### 5.2.2. 使用带有 nginx web 服务器的 PHP

本节论述了如何通过 **nginx** web 服务器运行 PHP 代码。

**先决条件**

- 在您的系统上安装 PHP 脚本语言。

**步骤**

1. 安装**nginx**软件包：

```
# dnf install nginx
```

2. 启动 **nginx** 服务器：

```
# systemctl start nginx
```

或者，如果 **nginx** 服务器已在您的系统中运行，请在安装 PHP 后重启 **nginx** 服务：

```
# systemctl restart nginx
```

3. 启动 **php-fpm** 服务：

```
# systemctl start php-fpm
```

4. 可选：在引导时启用这两个服务：

```
# systemctl enable php-fpm nginx
```

5. 要获取 PHP 设置的信息，请在 **/usr/share/nginx/html/** 目录中使用以下内容创建 **index.php** 文件：

```
echo '<?php phpinfo(); ?>' > /usr/share/nginx/html/index.php
```

6. 要运行 **index.php** 文件，请将浏览器指向：

```
http://<hostname>/
```

7. 可选：如果您有特定要求，请调整配置：
   - **/etc/nginx/nginx.conf** - **nginx** 主配置
   - **/etc/nginx/conf.d/php-fpm.conf** - FPM 配置 **nginx**
   - **/etc/php-fpm.conf** - FPM 主配置
   - **/etc/php-fpm.d/www.conf** - 默认 **www** 池配置

**例 5.2. 运行"Hello, World!" 使用 nginx 服务器的 PHP 脚本**

1. 在 **/usr/share/nginx/html/** 目录中为您的项目创建一个 **hello** 目录：

```
# mkdir hello
```

2. 在 **/usr/share/nginx/html/hello/** 目录中创建一个包含以下内容的 **hello.php** 文件：

```
# <!DOCTYPE html>
<html>
<head>
<title>Hello, World! Page</title>
</head>
<body>
<?php
   echo 'Hello, World!';
?>
</body>
</html>
```

3. 启动 **nginx** 服务器：

```
# systemctl start nginx
```

4. 要运行 **hello.php** 文件，请将浏览器指向：

```
http://<hostname>/hello/hello.php
```

因此，会显示带有 "Hello, World!" 文本的网页。

**其他资源**

- 设置和配置 NGINX

## 5.3. 使用命令行界面运行 PHP 脚本

PHP 脚本通常使用 Web 服务器运行，但也可以使用 命令行界面来运行。

**先决条件**

- 在您的系统上安装 PHP 脚本语言。

**步骤**

1. 在文本编辑器中，创建一个 ***filename*.php** 文件
将 *filename* 替换为您的文件名称。

2. 从命令行执行创建 ***filename*.php** 文件：

```
# php filename.php
```

**例 5.3. 运行"Hello, World!" 使用命令行界面 PHP 脚本**

1. 使用文本编辑器，创建包含以下内容的 **hello.php** 文件：

```
<?php
    echo 'Hello, World!';
?>
```

2. 从命令行执行 **hello.php** 文件：

```
# php hello.php
```

结果会输出 "Hello, World!"。

## 5.4. 其它资源

- **httpd(8)** - **httpd**服务的手册页，包含其命令行选项的完整列表。
- **httpd.conf(5)** - **httpd** 配置的 man page，描述 **httpd** 配置文件的结构和位置。
- **nginx(8)** - **nginx** web 服务器的 man page，其中包含其命令行选项的完整列表和信号列表。
- **php-fpm(8)** - PHP FPM 的 man page 描述其命令行选项和配置文件的完整列表。