# Red Hat Developer Tools 1

## 使用 Rust 1.66.1 Toolset

安装和使用 Rust 1.66.1 Toolset

Last Updated: 2023-08-18

# Red Hat Developer Tools 1 使用 Rust 1.66.1 Toolset

安装和使用 Rust 1.66.1 Toolset

Jacob Valdez
jvaldez@redhat.com

## 法律通告



## 摘要

Rust Toolset 是 Red Hat Enterprise Linux (RHEL)操作系统的开发人员的红帽产品。使用本指南概述 Rust Toolset， 了解如何调用和使用不同的 Rust 工具版本，并通过更深入的信息查找资源。

# 目录

# 使开源包含更多

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。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

# 第 1 章 RUST TOOLSET

Rust Toolset 是 Red Hat Enterprise Linux (RHEL)上的开发人员的产品。它为 Rust 编程语言、Rust 软件包管理器 每天、Rustfmt 格式工具和所需的库提供 **rustc** 编译器。

**Rust Toolset 作为 Red Hat Developer Tools for Red Hat Enterprise Linux 7 的一部分发布。对于 Red Hat Enterprise Linux 8，Rust Toolset 作为一个模块提供。Rust Toolset 作为 Red Hat Enterprise Linux 9 的软件包提供。**

## 1.1. RUST TOOLSET 组件

**以下组件作为 Rust Toolset 的一部分提供：**

| Name | 版本 | Description |
|---|---|---|
| **rust** | 1.66.1 | LLVM 的 Rust 编译器前端。 |
| **cargo** | 1.66.1 | Rust 的构建系统和依赖项管理器。 |
| **rustfmt** | 1.66.1 | 用于自动格式化 Rust 代码的工具。 |

## 1.2. RUST TOOLSET 兼容性

**在以下构架中为 Red Hat Enterprise Linux 7、Red Hat Enterprise Linux 8 和 Red Hat Enterprise Linux 9 提供 Rust Toolset：**

- **AMD 和 Intel 64 位**
- **64 位 ARM (RHEL 8 和 RHEL 9)**
- **IBM Power Systems, Little Endian**
- **IBM Power Systems，Mintle Endian （仅限 RHEL 7）**

- 64-bit IBM Z

## 1.3. 在 RED HAT ENTERPRISE LINUX 7 上访问 RUST TOOLSET

为了能够在 Red Hat Enterprise Linux 7 上安装 Rust Toolset，您必须访问并启用 Red Hat Developer Tools 和 Red Hat Software Collections 软件仓库。
如果这些软件仓库已附加到您的系统，请参阅安装 Rust Toolset。

**流程**

1. 运行以下命令安装 Wget ：

```
# yum install wget
```

2. 运行以下命令下载最新的订阅数据：

```
# subscription-manager refresh
```

3. 运行以下命令注册您的系统：

```
# subscription-manager register
```

要使用图形用户界面(GUI) 注册您的系统，请按照注册和取消注册系统 指南进行操作。

4. 显示所有可用订阅列表，并运行以下命令来识别池 ID：

```
# subscription-manager list --available
```

5. 在以池 ID 开头的行中找到 池 ID。

6. 运行以下命令，将提供对 Red Hat Developer Tools 存储库的访问权限的订阅附加：

```
# subscription-manager attach --pool=<pool ID from the subscription>
```

- 将 *subscription >* 中的 & lt; pool ID 替换为您在上一步中标识的池 ID。

7. 运行以下命令验证哪些订阅是否已附加到您的系统：

```
# sudo subscription-manager list --consumed
```

8. 运行以下命令启用 rhel-7-*变体*-devtools-rpms 存储库：

```
# subscription-manager repos --enable rhel-7-<variant>-devtools-rpms
```

- 将 *<variant >* 替换为您的 Red Hat Enterprise Linux 系统变体： server 或 workstation。

  使用 server 访问广泛的开发工具。

9. 运行以下命令启用 rhel-*variant*-RHSCL-7-rpms 存储库：

```
# subscription-manager repos --enable rhel-<variant>-rhscl-7-rpms
```

- 将 *<variant >* 替换为您的 Red Hat Enterprise Linux 系统变体： server 或 workstation。

10. 运行以下命令，在您的系统中添加 Red Hat Developer Tools GPG 密钥：

```
# cd /etc/pki/rpm-gpg
# wget -O RPM-GPG-KEY-redhat-devel https://www.redhat.com/security/data/a5787476.txt
# rpm --import RPM-GPG-KEY-redhat-devel
```

**其他资源**

- 有关注册您的系统并将其与订阅关联的更多信息，请参阅 红帽订阅管理 指南集合。

### 1.4. 安装 RUST TOOLSET

完成以下步骤，安装 Rust Toolset，包括所有开发和调试工具以及依赖软件包。请注意，Rust Toolset 依赖于 LLVM Toolset。

**先决条件**

- 在 Red Hat Enterprise Linux 7 中，提供对 Red Hat Developer Tools 内容集合的访问权限的订阅已附加到您的系统中。
  要附加订阅，请参阅 获取 Red Hat Enterprise Linux 7 中的 Rust Toolset 的访问权限。

- 已安装所有可用的 Red Hat Enterprise Linux 更新。

**流程**

在 Red Hat Enterprise Linux 7 中，运行以下命令安装 rust-toolset-1.66 集合：

```
# yum install rust-toolset-1.66
```

在 Red Hat Enterprise Linux 8 中，运行以下命令安装 rust-toolset 模块：

```
# yum module install rust-toolset
```

在 Red Hat Enterprise Linux 9 中，运行以下命令安装 rust-toolset 软件包：

```
# dnf install rust-toolset
```

### 1.5. 安装 RUST 文档

*Rust 编程语言书* 可作为可安装的文档提供。

**先决条件**

- 已安装 Rust Toolset。
  如需更多信息，请参阅安装 Rust Toolset。

**流程**

要安装 rust-doc 软件包，请运行以下命令：

- 在 Red Hat Enterprise Linux 7 上：

```
# yum install rust-toolset-1.66-rust-doc
```

您可以在以下路径中找到 *Rust 编程语言* 书：/opt/rh/rust-toolset-1.66/root/usr/share/doc/rust/html/index.html。
您可以在以下路径中找到所有 Rust 代码软件包的 API 文档：/opt/rh/rust-toolset-1.66/root/usr/share/doc/rust/html/std/index.html。

- 在 Red Hat Enterprise Linux 8 中：

```
# yum install rust-doc
```

您可以在以下路径下找到 *Rust 编程语言* 书：/usr/share/doc/rust/html/index.html。
您可以在以下路径中找到所有 Rust 代码软件包的 API 文档：/usr/share/doc/rust/html/std/index.html。

- 在 Red Hat Enterprise Linux 9 中：

```
# dnf install rust-doc
```

您可以在以下路径下找到 *Rust 编程语言* 书：/usr/share/doc/rust/html/index.html。
您可以在以下路径中找到所有 Rust 代码软件包的 API 文档：/usr/share/doc/rust/html/std/index.html。

### 1.6. 安装 THEGO 文档

Rust *go, Rust 的 Package Manager* 书可作为 Icego 的可安装文档提供。

注意

在 Rust Toolset 1.66 中，cargo-doc 软件包包含在 rust-doc 软件包中。

先决条件

- 已安装 Rust Toolset。
如需更多信息，请参阅安装 Rust Toolset。

**流程**

- **要安装 cargo-doc 软件包，请运行：**
  - **在 Red Hat Enterprise Linux 7 上：**

```
# yum install rust-toolset-1.66-cargo-doc
```

**您可以在以下路径下找到 define *go 书*： /opt/rh/rust-toolset-1.66/root/usr/share/doc/cargo/html/index.html。**

  - **在 Red Hat Enterprise Linux 8 中：**

```
# yum install cargo-doc
```

**您可以在以下路径中找到 *Rustgo, Rust 的软件包管理器* 书：/usr/share/doc/cargo/html/index.html。**

  - **在 Red Hat Enterprise Linux 9 中：**

```
# dnf install cargo-doc
```

**您可以在以下路径中找到 *Rustgo, Rust 的软件包管理器* 书：/usr/share/doc/cargo/html/index.html。**

## 1.7. 其他资源

- **有关 Rust 编程语言的更多信息，请参阅 官方 Rust 文档。**

# 第 2 章 LIBPMEMGO 构建工具

cargo 是 Rust 编译器 rustc 的构建工具和前端，以及软件包和依赖项管理器。它允许 Rust 项目声明特定版本要求的依赖项，解决所有依赖项图、下载软件包和构建以及测试整个项目。

Rust Toolset 提供了 slirpgo 1.66.1。

## 2.1. NAVIGATEGO 目录结构和文件放置

libpmemgo 构建工具使用 set 约定来定义在 thego 软件包中定义目录结构和文件放置。运行 cargo new 命令会为清单和项目文件生成软件包目录结构和模板。默认情况下，它还在软件包根目录中初始化一个新的 Git 存储库。

对于二进制程序，sego 会创建一个目录 *project_name*，其中包含一个名为 called go.toml 的文本文件，另一个子目录 src 包含名为 main.rs 的文本文件。

### 其他资源

- 有关 definego 目录结构的更多信息，请参阅 知识库文章 - Package Layout。
- 有关 Rust 代码机构的信息，请参阅 Rust 编程语言 - 使用 Packages、Crates 和 Modules 管理 Growing 项目。

## 2.2. 创建 RUST 项目

创建一个新的 Rust 项目，它根据 definego 约定设置。有关 Applego 约定的更多信息，请参阅 the go 目录结构和文件放置。

### 流程

运行以下命令来创建 Rust 项目：

- 在 Red Hat Enterprise Linux 7 上：

```
$ scl enable rust-toolset-1.66 'cargo new --bin <project_name>'
```

- 将 *<project_name>* 替换为您的项目名称。

- 在 Red Hat Enterprise Linux 8 中：

```
$ cargo new --bin <project_name>
```

  - 将 *<project_name>* 替换为您的项目名称。

- 在 Red Hat Enterprise Linux 9 中：

```
$ cargo new --bin <project_name>
```

  - 将 *<project_name>* 替换为您的项目名称。

**注意**

若要编辑项目代码，请编辑主可执行文件 main.rs，并将新源文件添加到 src 子目录中。

**其他资源**

- 有关配置项目并添加依赖项的详情，请参考 配置 Rust 项目依赖项。

### 2.3. 创建 RUST 库项目

完成以下步骤，使用 libpmemgo 构建工具创建 Rust 库项目。

**流程**

要创建 Rust 库项目，请运行以下命令：

- 在 Red Hat Enterprise Linux 7 上：

```
$ scl enable rust-toolset-1.66 'cargo new --lib <project_name>'
```

- 将 *<project_name>* 替换为 Rust 项目的名称。

- 在 Red Hat Enterprise Linux 8 中：

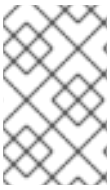
```
$ cargo new --lib <project_name>
```

  - 将 *<project_name>* 替换为 Rust 项目的名称。

- 在 Red Hat Enterprise Linux 9 中：

```
$ cargo new --lib <project_name>
```

  - 将 *<project_name>* 替换为 Rust 项目的名称。



**注意**

若要编辑项目代码，请编辑 src 子目录中的源代码 lib.rs。

**其他资源**

- 使用软件包、校准和模块管理 Growing 项目

### 2.4. 构建 RUST 项目

使用 slirpgo 构建工具构建 Rust 项目。cargo 解决了项目的所有依赖项，下载缺少的依赖项，并使用 rustc 编译器编译它。

默认情况下，项目以 debug 模式构建并编译。有关以发行版本模式编译项目的详情，请参考在 发行版本模式中构建 Rust 项目。

**先决条件**

- 一个现有的 Rust 项目。
有关如何创建 Rust 项目的详情，请参考 创建 Rust 项目。

流程

- 要构建 consistgo 管理的 Rust 项目，请在项目目录中运行：
  - 在 Red Hat Enterprise Linux 7 上：

```
$ scl enable rust-toolset-1.66 'cargo build'
```

  - 在 Red Hat Enterprise Linux 8 中：

```
$ cargo build
```

  - 在 Red Hat Enterprise Linux 9 中：

```
$ cargo build
```

- 要在不需要构建可执行文件时验证 Rust 程序是否可以构建，请运行：

```
$ cargo check
```

### 2.5. 以发行版本模式构建 RUST 项目

以发行模式使用 theubrgo 构建工具构建 Rust 项目。发行版本模式优化源代码，因此可以增加编译时间，同时确保编译的二进制文件可以更快地运行。使用此模式生成适合发行和生产的优化的工件。cargo 解决了项目的所有依赖项，下载缺少的依赖项，并使用 rustc 编译器编译它。

有关以 debug 模式编译项目的详情，请参考 构建 Rust 项目。

先决条件

- 一个现有的 Rust 项目。
  有关如何创建 Rust 项目的详情，请参考 创建 Rust 项目。

流程

- 要以发行版本模式构建项目，请运行：

- 在 Red Hat Enterprise Linux 7 上：

```
$ scl enable rust-toolset-1.66 'cargo build --release'
```

- 在 Red Hat Enterprise Linux 8 中：

```
$ cargo build --release
```

- 在 Red Hat Enterprise Linux 9 中：

```
$ cargo build --release
```

- 要在不需要构建可执行文件时验证 Rust 程序是否可以构建，请运行：

```
$ cargo check
```

### 2.6. 运行 RUST 程序

使用 libpmemgo 构建工具运行 Rust 项目。cargo 首先重建您的项目，然后运行生成的可执行文件。如果在开发过程中使用，cargo run 命令可以正确地独立于构建模式解析输出路径。

**先决条件**

- 构建的 Rust 项目。
  有关如何构建 Rust 项目的详情，请参考 构建 Rust 项目。

**流程**

要运行 Rust 程序作为项目管理，请在项目目录中运行：

- 在 Red Hat Enterprise Linux 7 上：

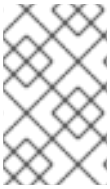
```
$ scl enable rust-toolset-1.66 'cargo run'
```

- 在 Red Hat Enterprise Linux 8 中：

```
$ cargo run
```

- 在 Red Hat Enterprise Linux 9 中：

```
$ cargo run
```



**注意**

如果您的程序尚未构建，Cisd 会在运行它前构建您的程序。

## 2.7. 测试 RUST 项目

使用 guestfishgo 构建工具测试您的 Rust 程序。cargo 首先重建您的项目，然后运行项目中找到的测试。请注意，您只能测试可用的、货币性的功能，且不用任何参数。功能返回类型必须是 （） 或 Result< （），其中 E: Error。

默认情况下，R Rust 项目以 debug 模式进行测试。有关以发行版本模式测试项目的详情，请参考在发行版本模式中测试 Rust 项目。

### 先决条件

- 构建的 Rust 项目。
有关如何构建 Rust 项目的详情，请参考 构建 Rust 项目。

### 流程

- 在功能前面的 前添加 test 属性 192.168.1.0/24[test]。
- 要对 consistgo 管理的 Rust 项目运行测试，请在项目目录中运行：
  - 在 Red Hat Enterprise Linux 7 上：

```
$ scl enable rust-toolset-1.66 'cargo test'
```

  - 在 Red Hat Enterprise Linux 8 中：

-

```
$ cargo test
```

- 在 Red Hat Enterprise Linux 9 中：

```
$ cargo test
```

**其他资源**

- 有关在 Rust 项目中执行测试的更多信息，请参阅 Rust Reference - 测试属性。

### 2.8. 以发行版本模式测试 RUST 项目

使用 guestfishgo 构建工具以发行版本模式测试您的 Rust 程序。发行版本模式优化源代码，因此可以增加编译时间，同时确保编译的二进制文件可以更快地运行。使用此模式生成适合发行和生产的优化的工件。
cargo 首先重建您的项目，然后运行项目中找到的测试。请注意，您只能测试可用的、货币性的功能，且不用任何参数。功能返回类型必须是 （） 或 Result< （），其中 E: Error。

有关以 debug 模式测试项目的详情，请参考 测试 Rust 项目。

**先决条件**

- 构建的 Rust 项目。
有关如何构建 Rust 项目的详情，请参考 构建 Rust 项目。

**流程**

- 在功能前面的 前添加 test 属性 192.168.1.0/24[test]。

- 要在发行版本模式中对 consistgo 管理的 Rust 项目运行测试，请在项目目录中运行：

  - 在 Red Hat Enterprise Linux 7 上：

```
$ scl enable rust-toolset-1.66 'cargo test --release'
```

  - 在 Red Hat Enterprise Linux 8 中：

```
$ cargo test --release
```

- 在 Red Hat Enterprise Linux 9 中：

```
$ cargo test --release
```

**其他资源**

- 有关在 Rust 项目中执行测试的更多信息，请参阅 Rust Reference - 测试属性。

## 2.9. 配置 RUST 项目依赖项

使用 guestfishgo 构建工具配置 Rust 项目的依赖项。若要指定由 definego 管理的项目的依赖项，请编辑项目目录中的文件，并重建项目。cargo 下载 Rust 代码软件包及其依赖项，将其存储在本地，构建所有项目源代码，包括依赖项代码软件包，并运行生成的可执行文件。

**先决条件**

- 构建的 Rust 项目。
有关如何构建 Rust 项目的详情，请参考 构建 Rust 项目。

**流程**

1. 在项目目录中，打开文件 192.168.0.go .toml。

2. 移到标有 [dependencies] 的部分。
每个依赖项都以以下格式在新行中列出：

```
crate_name = version
```

Rust 代码软件包称为 crates。

3. 编辑依赖项。

4. 运行以下命令重建项目：

- **在 Red Hat Enterprise Linux 7 上：**

```
$ scl enable rust-toolset-1.66 'cargo build'
```

- **在 Red Hat Enterprise Linux 8 中：**

```
$ cargo build
```

- **在 Red Hat Enterprise Linux 9 中：**

```
$ cargo build
```

5. **使用以下命令运行项目：**

- **在 Red Hat Enterprise Linux 7 上：**

```
$ scl enable rust-toolset-1.66 'cargo run'
```

- **在 Red Hat Enterprise Linux 8 中：**

```
$ cargo run
```

- **在 Red Hat Enterprise Linux 9 中：**

```
$ cargo run
```

**其他资源**

- **有关配置 Rust 依赖项的更多信息，请参阅 define go Book - 指定依赖项。**

### 2.10. 为 RUST 项目构建文档

**使用 definego 工具，从源代码中的注释生成文档，这些文档标记为提取。请注意，文档注释只针对公共功能、变量和成员提取。**

**先决条件**

- 构建的 Rust 项目。
有关如何构建 Rust 项目的详情，请参考 构建 Rust 项目。

- 配置的依赖项。
有关配置依赖项的更多信息，请参阅配置 Rust 项目依赖项。

**流程**

- 要标记提取的注释，请使用三个斜杠 ///，并将您的注释放在其记录的行的开头。
cargo 支持您的评论的 Markdown 语言。

- 要使用 definego 构建项目文档，请在项目目录中运行：

  - 在 Red Hat Enterprise Linux 7 上：

```
$ scl enable rust-toolset-1.66 'cargo doc --no-deps'
```

  - 在 Red Hat Enterprise Linux 8 中：

```
$ cargo doc --no-deps
```

  - 在 Red Hat Enterprise Linux 9 中：

```
$ cargo doc --no-deps
```

生成的文档位于 .target/doc 目录中。

**其他资源**

- 有关使用 guestfishgo 构建文档的更多信息，请参阅 Rust 编程语言 - 使用文档注释。

## 2.11. 在 RED HAT ENTERPRISE LINUX 8 和 RED HAT ENTERPRISE LINUX 9 BETA 中使用 RUST 将代码编译到 WEBASSEMBLY 二进制文件中

完成以下步骤以安装 WebAssembly 标准库。

**先决条件**

- 已安装 Rust Toolset。
  如需更多信息，请参阅安装 Rust Toolset。

**流程**

- 要安装 WebAssembly 标准库，请运行：
  - 在 Red Hat Enterprise Linux 8 中：

```
# yum install rust-std-static-wasm32-unknown-unknown
```

  - 在 Red Hat Enterprise Linux 9 中：

```
# dnf install rust-std-static-wasm32-unknown-unknown
```

- 要将 WebAssembly 与 definego 搭配使用，请运行：
  - 在 Red Hat Enterprise Linux 8 中：

```
# cargo <command> --target wasm32-unknown-unknown
```

    将 <command > 替换为您要运行的 mailboxgo 命令。

  - 在 Red Hat Enterprise Linux 9 中：

```
# cargo <command> --target wasm32-unknown-unknown
```

    将 <command > 替换为您要运行的 mailboxgo 命令。

**其他资源**

- 有关 WebAssembly 的更多信息，请参阅官方 Rust 和 WebAssembly 文档或 Rust 和 WebAssembly 书。

## 2.12. 供应商 RUST 项目依赖项

创建 Rust 项目的依赖项的本地副本，以使用 libpmemgo 构建工具离线重新发布和重复使用。此流程被称为厂商项目依赖项。供应商的依赖项包括在 Windows 操作系统上构建项目的 Rust 代码软件包位于 厂商 目录中。厂商的依赖关系可在不连接互联网的情况下使用 vendored dependencies。

### 先决条件

- 构建的 Rust 项目。
有关如何构建 Rust 项目的详情，请参考 构建 Rust 项目。

- 配置的依赖项。
有关配置依赖项的更多信息，请参阅配置 Rust 项目依赖项。

### 流程

要使用 definego 为 Rust 项目提供依赖项，请在项目目录中运行：

- 在 Red Hat Enterprise Linux 7 上：

```
$ scl enable rust-toolset-1.66 'cargo vendor'
```

- 在 Red Hat Enterprise Linux 8 中：

```
$ cargo vendor
```

- 在 Red Hat Enterprise Linux 9 中：

```
$ cargo vendor
```

## 2.13. 其他资源

- 有关 Dango 的更多信息，请参阅 Official go 指南。

- **要显示 Rust Toolset 中包含的手册页，请运行：**
  - 对于 **Red Hat Enterprise Linux 7**：

```
$ scl enable rust-toolset-1.66 'man cargo'
```

  - 对于 **Red Hat Enterprise Linux 8**：

```
$ man cargo
```

  - 对于 **Red Hat Enterprise Linux 9**：

```
$ man cargo
```

# 第 3 章 RUSTFMT 格式工具

使用 rustfmt 格式工具，您可以自动格式化 Rust 程序的源代码。您可以将 rusftmt 用作独立工具，或通过 gego 使用。

## 3.1. 安装 RUSTFMT

完成以下步骤以安装 rustfmt 格式工具。

### 先决条件

- 已安装 Rust Toolset。
  如需更多信息，请参阅安装 Rust Toolset。

### 流程

运行以下命令来安装 rustfmt ：

- 在 Red Hat Enterprise Linux 7 上：

```
# yum install rust-toolset-1.66-rustfmt
```

- 在 Red Hat Enterprise Linux 8 中：

```
# yum install rustfmt
```

- 在 Red Hat Enterprise Linux 9 中：

```
# dnf install rustfmt
```

## 3.2. 使用 RUSTFMT 作为独立工具

使用 rustfmt 作为独立工具格式化 Rust 源文件及其所有依赖项。作为替代方案，将 rustfmt 与 guestfishgo 构建工具一起使用。如需更多信息，请参阅使用带有 definego 的 rustfmt。

### 先决条件

- 一个现有的 Rust 项目。

有关如何创建 Rust 项目的详情，请参考 创建 Rust 项目。

**流程**

要使用 rustfmt 作为独立工具格式化 Rust 源文件，请运行以下命令：

- 在 Red Hat Enterprise Linux 7 上：

```
$ scl enable rust-toolset-1.66 'rustfmt <source-file>'
```

  - 将 *<source_file>* 替换为源文件的名称。

或者，您可以将 <*source_file*> 替换为标准输入。然后，tairfmt 在标准输出中提供其输出。

- 在 Red Hat Enterprise Linux 8 中：

```
$ rustfmt <source-file>
```

  - 将 *<source_file>* 替换为源文件的名称。

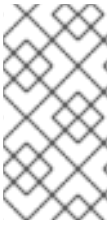
或者，您可以将 <*source_file*> 替换为标准输入。然后，tairfmt 在标准输出中提供其输出。

- 在 Red Hat Enterprise Linux 9 中：

```
$ rustfmt <source-file>
```

  - 将 *<source_file>* 替换为源文件的名称。

或者，您可以将 <*source_file*> 替换为标准输入。然后，tairfmt 在标准输出中提供其输出。



**注意**

默认情况下，lib air fmt 修改受影响的文件，而不显示详情或创建备份。要显示详情并创建备份，请运行带有 --write-mode *值的* rustfmt。

### 3.3. 在 GUESTFISHGO 构建工具中使用 RUSTFMT

将 rustfmt 工具与 guestfishgo 搭配使用，以格式化 Rust 源文件及其所有依赖项。
另外，使用 rustfmt 作为独立工具。如需更多信息，请参阅使用 rustfmt 作为独立工具。

**先决条件**

- 一个现有的 Rust 项目。
  有关如何创建 Rust 项目的详情，请参考 创建 Rust 项目。

**流程**

要格式化 thego 代码软件包中的所有源文件，请运行以下命令：

- 在 Red Hat Enterprise Linux 7 上：

```
$ scl enable rust-toolset-1.66 'cargo fmt'
```

- 在 Red Hat Enterprise Linux 8 中：

```
$ cargo fmt
```

- 在 Red Hat Enterprise Linux 9 中：

```
$ cargo fmt
```

**注意**

要更改 rustfmt 格式选项，请在项目目录中创建配置文件 rustfmt.toml，并将您的配置添加到该文件中。

### 3.4. 其他资源

- 要显示 rustfmt 的帮助页面，请运行：
  - 在 Red Hat Enterprise Linux 7 上：

```
$ scl enable rust-toolset-1.66 'rustfmt --help'
```

- 在 Red Hat Enterprise Linux 8 中：

```
$ rustfmt --help
```

- 在 Red Hat Enterprise Linux 9 中：

```
$ rustfmt --help
```

- 要配置 rustfmt 工具，请编辑文件 Configuration .md。
  - 在 Red Hat Enterprise Linux 7 中，您可以在以下路径中找到：

    /opt/rh/rust-toolset-1.66/root/usr/share/doc/rust-toolset-1.66-rustfmt-1.66.1/Configurations.md

  - 在 Red Hat Enterprise Linux 8 中，您可以在以下路径中找到：

    /usr/share/doc/rustfmt/Configurations.md

  - 在 Red Hat Enterprise Linux 9 中，您可以在以下路径中找到：

    /usr/share/doc/rustfmt/Configurations.md

# 第 4 章 在 RHEL 8 上使用 RUST TOOLSET 的容器镜像

在 RHEL 8 中，您可以使用 Containerfiles 在 Red Hat Universal Base Images (UBI)容器之上构建自己的 Rust Toolset 容器镜像。

## 4.1. 在 RHEL 8 上创建 RUST TOOLSET 的容器镜像

在 RHEL 8 中，Rust Toolset 软件包是 Red Hat Universal Base Images (UBI)存储库的一部分。要保持容器大小小，请只安装单独的软件包而不是整个 Rust Toolset。

**先决条件**

- 现有的 Containerfile。
  有关创建 Containerfiles 的更多信息，请参阅 Dockerfile 参考 页面。

**流程**

- 访问 红帽容器目录。
- 选择 UBI。
- 点 Get this image 并按照说明进行操作。
- 要创建包含 Rust Toolset 的容器，请在 Containerfile 中添加以下行：

```
FROM registry.access.redhat.com/ubi8/ubi:latest

RUN yum install -y rust-toolset
```

- 要创建仅包含单个软件包的容器镜像，请在 Containerfile 中添加以下行：

```
RUN yum install <package-name>
```

- 将 < *package_name* > 替换为您要安装的软件包的名称。

### 4.2. 其他资源

- 有关 Red Hat UBI 镜像的更多信息，请参阅使用容器镜像。

- 有关 Red Hat UBI 存储库的更多信息，请参阅 通用基础镜像(UBI)：镜像、存储库、软件包和源代码。

# 第 5 章 RUST 1.66.1 TOOLSET 中的更改

Rust Toolset 已从 RHEL 7、RHEL 8 和 RHEL 9 版本 1.62.1 更新至 1.66.1。

主要变更包括：

- thread::scope API 创建一个字典范围，其中本地变量可以被新生成的线程安全地借用，且这些线程都保证在范围结束前退出。
- hint::black_box API 向编译器优化添加了一个障碍，这对于保留基准中的行为很有用，否则这些行为可能会被优化掉。
- .await 关键字现在使用 IntoFuture 特征进行转换，类似于 for 和 IntoIterator 之间的关系。
- 通用关联类型(GAT)允许特征包含具有通用参数的类型别名，对类型和生命周期启用新抽象。
- 新的 let-else 语句允许本地变量与条件模式匹配绑定，在模式不匹配时执行分支 else 块。
- 标记的块允许 break 语句跳到块的末尾，可选包括表达式值。
- Rust -analyzer 是语言服务器协议的新实现，在很多编辑器中启用 Rust 支持。这取代了以前的 rls 软件包，但您可能需要调整编辑器配置以迁移到 rust-analyzer。
- cargo 有一个新的 cargo remove 子命令，用于从 libpmemgo .toml 中删除依赖项。

有关更新的详情，请查看上游发布公告系列：

- 宣布 Rust 1.63.0。
- 宣布 Rust 1.64.0.

- 宣布 Rust 1.65.0。
- 宣布 Rust 1.66.0。