Red Hat Developer Toolset 11

用户指南

安装和使用 Red Hat Developer Toolset

Last Updated: 2022-07-31

Red Hat Developer Toolset 11 用户指南

安装和使用 Red Hat Developer Toolset

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

Red Hat Developer Toolset 是 Red Hat Enterprise Linux 平台上的开发人员提供的红帽产品。Red Hat Developer Toolset User 指南提供了这个产品的概述，解释了如何调用和使用 Red Hat Developer Toolset 版本，以及链接到包含更深入的信息的资源的链接。

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。

部分 I. 简介

第 1 章 RED HAT DEVELOPER TOOLSET

1.1. 关于 RED HAT DEVELOPER TOOLSET

Red Hat Developer Toolset是 Red Hat Enterprise Linux 平台上的开发人员提供的红帽产品。它提供了一组完整的开发和性能分析工具，可用于多个版本的 Red Hat Enterprise Linux。然后，也可以使用 Red Hat Developer Toolset 工具chain 构建的可执行文件，也可以在多个 Red Hat Enterprise Linux 版本中运行。有关兼容性信息，请参阅 第 1.3 节 “兼容性”。

当在这些平台中安装 Red Hat Enterprise Linux 7 时，Red Hat Developer Toolset 不会替换 Red Hat Enterprise Linux 7 提供的默认系统工具。相反，一组并行的开发人员工具提供了一个替代的，这些工具的新版本供开发人员使用。例如，默认的编译器和调试器保留基础 Red Hat Enterprise Linux 系统提供的。

Red Hat Developer Toolset 11.0 中的新功能

从 Red Hat Developer Toolset 4.1 开始，Red Hat Developer Toolset 内容也以 ISO 格式以及位于 https://access.redhat.com/downloads 的 Red Hat Software Collections 内容（特别是 Server 和 Workstation ）中的内容一起提供。请注意，需要 *Optional* 频道（在 第 1.5.3 节 “安装可选软件包” 中讨论）的软件包无法从 ISO 镜像安装。

表 1.1. Red Hat Developer Toolset 组件

Name	版本	描述
GCC	11.2	可移植的编译器套件，支持 C、C++ 和 Fortran。
binutils	2.36	用于检查和操作对象文件和二进制文件的二进制工具和其他实用程序的集合。
elfutils	0.185	用于检查和操作 ELF 文件的二进制工具和其他实用程序的集合。
dwz	0.14	用于优化 ELF 共享库和 ELF 可执行文件大小中包含的 DWARF 调试信息的工具。
GDB	10.2	用于 C、C++ 和 Fortran 中编写的程序的命令行调试器。
ltrace	0.7.91	用于显示程序发出的动态库调用的调试工具。它还可以监控程序执行的系统调用。
strace	5.13	一个调试工具，用于监控程序使用它的系统调用和它收到的信号。
memstomp	0.1.5	一个调试工具，用于识别对带有不同标准不允许重叠的内存区域的库函数调用。
SystemTap	4.5	追踪和探测工具，可监控整个系统的活动，而无需检测、重新编译、安装和重新启动。
Valgrind	3.17.0	工具框架和一些用于分析应用程序的工具，以检测内存错误、识别内存管理问题，并报告系统调用中不正确的参数的使用。

Name	版本	描述
OProfile	1.4.0	使用处理器中性能监控硬件的系统范围配置集来检索系统中内核和可执行文件的信息。
Dyninst	11.0.0	在执行期间使用用户空间可执行文件的库。
make	4.3	依赖项跟踪构建自动化工具。
annobin	9.82	构建安全检查工具。

Red Hat Developer Toolset 与 "技术预览" 在 Red Hat Enterprise Linux 中以前提供的，两个重要方面有所不同：

1. Red Hat Developer Toolset 可用于 Red Hat Enterprise Linux 的多个主版本和次版本，详情请参考 第 1.3 节 “兼容性”。

2. 与早期 Red Hat Enterprise Linux 提供的技术预览编译器和其他工具不同，Red Hat Developer Toolset 在 Red Hat Enterprise Linux 订阅级别协议中被完全支持，它完全受支持。

从每个主版本的发行版本发行两年内，会向 Red Hat Developer Toolset 订阅者发布重要的程序错误修复和安全勘误。Red Hat Developer Toolset 的新主要版本会每年发布，为现有组件和添加新组件提供显著更新。一个次发行版本，在每个新的主发行版本发布后发布 6 个月，都有对程序错误修复、安全勘误和新的次版本的更小的更新。

另外，Red Hat Enterprise Linux 应用程序兼容性规格 也适用于 Red Hat Developer Toolset（受使用较新的 C++11 语言功能的一些限制，在 第 2.2.4 节 “c++ 兼容性”中详述）。

重要

Red Hat Developer Toolset 提供的应用程序和库不会替换 Red Hat Enterprise Linux 系统版本，也不会首选使用系统版本。使用名为 **Software Collections** 的框架，一组额外的开发人员工具安装到 **/opt/** 目录中，由用户使用 **scl** 程序根据需要进行显式启用。

1.2. 主要功能

Red Hat Developer Toolset 11.0 有以下更改：

- Red Hat Developer Toolset 版本 **GNU Compiler Collection**（**GCC**）已升级至版本 11.2，它带有许多新功能和程序错误修复。

- Red Hat Developer Toolset 版本 **GNU Debugger**（**GDB**）已升级至版本 10.2，它带有很多新功能和程序错误修复。

有关本发行版本中引入的更改和功能的完整列表，请参阅 附录 A, *版本 11.0 的更改*。

1.3. 兼容性

对于多个架构，Red Hat Enterprise Linux 7 提供了 Red Hat Developer Toolset 11.0。

有关 ABI 兼容性信息，请参阅 第 2.2.4 节 “c++ 兼容性”。

表 1.2. Red Hat Developer Toolset 11.0 兼容性

	在 Red Hat Enterprise Linux 7.7 上运行	在 Red Hat Enterprise Linux 7.9 上运行
使用 Red Hat Enterprise Linux 7.7 构建	支持	支持
使用 Red Hat Enterprise Linux 7.9 构建	不支持	支持

架构支持
Red Hat Developer Toolset 在以下构架中提供：

- 64 位 Intel 和 AMD 构架
- IBM Power Systems，big endian
- IBM Power Systems, little endian
- 64-bit IBM Z

1.4. 获取访问 RED HAT DEVELOPER TOOLSET

Red Hat Developer Toolset 是作为 Red Hat Software Collections 的一部分提供的。

此内容集可供列出的有 Red Hat Enterprise Linux 7 订阅的客户使用 https://access.redhat.com/solutions/472793。

使用 Red Hat Subscription Management 启用 Red Hat Developer Toolset。有关如何将系统注册到此订阅管理服务的详情，请查看 Red Hat Subscription Management 指南。

1.4.1. 使用 Red Hat Software Collections

完成以下步骤以附加可访问 Red Hat Software Collections（包括 Red Hat Developer Toolset）软件仓库的订阅，然后启用该软件仓库：

1. 确定提供 Red Hat Software Collections（以及 Red Hat Developer Toolset）的订阅池 ID。要做到这一点，显示适合您的系统的所有订阅列表：

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

对于每个可用的订阅，这个命令会显示其名称、唯一标识符、到期日期和其他与您的订阅相关的详情。池 ID 在以 **池 ID** 开头的行中列出。

有关提供对 Red Hat Developer Toolset 访问权限的订阅的完整列表，请参阅 https://access.redhat.com/solutions/472793。

2. 将适当的订阅附加到您的系统：

```
# subscription-manager attach --pool=pool_id
```

使用您在上一步中确定的池 ID 替换 *pool_id*。在任何时候验证您的系统当前附加的订阅列表：

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

3. 确定 Red Hat Software Collections 存储库的确切名称。检索存储库元数据，并显示可用的 Yum 存储库列表：

```
# subscription-manager repos --list
```

存储库名称取决于您使用的 Red Hat Enterprise Linux 的特定版本，其格式如下：

```
rhel-variant-rhscl-version-rpms
rhel-variant-rhscl-version-debug-rpms
rhel-variant-rhscl-version-source-rpms
```

另外，某些软件包，如 **devtoolset-11-gcc-plugin-devel**，它依赖于 *可选* 频道中仅提供的软件包。使用这些软件包的软件仓库名称采用以下格式：

```
rhel-version-variant-optional-rpms
rhel-version-variant-optional-debug-rpms
rhel-version-variant-optional-source-rpms
```

对于常规软件仓库和可选软件仓库，请使用 Red Hat Enterprise Linux 系统变体（**服务器或工作站**）替换变体，并使用 Red Hat Enterprise Linux 系统版本 **7** 替换版本。

4. *从第 3 步启用存储库。3：*

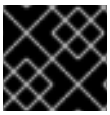
```
# subscription-manager repos --enable repository
```

使用要启用的存储库的名称替换 repository。

订阅附加到系统后，您可以安装 Red Hat Developer Toolset，如 第 1.5 节 "安装 Red Hat Developer Toolset" 所述。有关如何使用红帽订阅管理注册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。

1.5. 安装 RED HAT DEVELOPER TOOLSET

Red Hat Developer Toolset 作为 RPM 软件包的集合发布，它们通过使用 Red Hat Enterprise Linux 中包含的标准软件包管理工具进行安装、更新、卸载和检查。请注意，需要一个有效的订阅，提供对 Red Hat Software Collections 内容集的访问权限，以便在您的系统中安装 Red Hat Developer Toolset。有关如何将您的系统与适当的订阅相关联并可以访问 Red Hat Developer Toolset 的详情，请参考 第 1.4 节 "获取访问 Red Hat Developer Toolset"。



重要

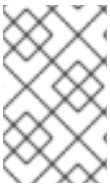
在安装 Red Hat Developer Toolset 前，安装所有可用的 Red Hat Enterprise Linux 更新。

1.5.1. 安装所有可用的组件

*要安装 Red Hat Developer Toolset 中的所有组件，请安装 **devtoolset-11** 软件包：*

```
# yum install devtoolset-11
```

这会在系统中安装所有开发、调试和性能监控工具，以及其它依赖软件包。另外，您可以选择只安装所选软件包组，如 第 1.5.2 节 "安装单个软件包组" 所述。



注意

请注意，自 Red Hat Developer Toolset 3.0 开始，**scl-utils** 软件包不是 Red Hat Developer Toolset 的一部分，它是之前版本的更改，**scl** 工具程序与 Red Hat Developer Toolset 软件集合一起安装。

1.5.2. 安装单个软件包组

为了便于安装某些组件，如集成的开发环境或软件开发工具链，Red Hat Developer Toolset 使用多个 meta 软件包发布，可让您安装所选软件包组，如 表 1.3 “Red Hat Developer Toolset Meta Packages” 所述。

表 1.3. Red Hat Developer Toolset Meta Packages

软件包名称	描述	安装的组件
devtoolset-11-perftools	性能监控工具	systemtap, Valgrind, OProfile, Dyninst
devtoolset-11-toolchain	开发和调试工具	GCC, make, GDB, binutils, elfutils, dwz, memstomp, strace, ltrace

安装任何这些 meta 软件包：

```
# yum install package_name
```

使用您要安装的 meta 软件包的空格分隔列表替换 package_name。例如，只安装开发和调试工具链以及依赖于它的软件包：

```
# yum install devtoolset-11-toolchain
```

另外，您可以选择安装所有可用组件，如 第 1.5.1 节 “安装所有可用的组件” 所述。

1.5.3. 安装可选软件包

Red Hat Developer Toolset 带有很多没有默认安装的可选软件包。要列出所有可用的 Red Hat Developer Toolset 软件包，但还没有安装在系统中：

```
$ yum list available devtoolset-11-\*
```

安装任何这些可选软件包：

```
# yum install package_name
```

使用您要安装的软件包列表替换 package_name。例如，安装 **devtoolset-11-gdb-gdbserver** 和 **devtoolset-11-gdb-doc** 软件包：

```
# yum install devtoolset-11-gdb-gdbserver devtoolset-11-gdb-doc
```

1.5.4. 安装调试信息

要为任何 Red Hat Developer Toolset 软件包安装调试信息，请确保已安装并运行 **yum-utils** 软件包：

```
# debuginfo-install package_name
```

例如，要为 **devtoolset-11-dwz** 软件包安装调试信息：

```
# debuginfo-install devtoolset-11-dwz
```

请注意，要使用这个命令，您需要使用这些软件包访问存储库。如果您的系统使用 Red Hat Subscription Management 注册，请启用 **rhel-variant-rhscl-version-debug-rpms** 软件仓库，如 第 1.4 节 "获取访问 Red Hat Developer Toolset" 所述。有关如何访问 debuginfo 软件包的更多信息，请参阅 https://access.redhat.com/site/solutions/9907。

注意

devtoolset-11-package_name-debuginfo 软件包可能会与基础 Red Hat Enterprise Linux 系统或其他版本的 Red Hat Developer Toolset 中对应的软件包冲突。在多 lib 环境中也会发生这个冲突，其中 64 位 debuginfo 软件包与 32 位 debuginfo 软件包有冲突。

在安装 Red Hat Developer Toolset 11.0 前手动卸载冲突的debuginfo 软件包，并在需要时只安装相关的调试信息软件包。

1.6. 更新 RED HAT DEVELOPER TOOLSET

1.6.1. 更新至次版本

当有新的 Red Hat Developer Toolset 的次版本时，更新您的 Red Hat Enterprise Linux 安装：

```
# yum update
```

这会更新 Red Hat Enterprise Linux 系统中的所有软件包，包括开发、调试和性能监控工具的 Red Hat Developer Toolset 版本，以及其他依赖的软件包。

重要

使用 Red Hat Developer Toolset 需要删除任何早期的预发布版本。另外，从 Red Hat Developer Toolset 的预发行版本（包括 beta 版本）无法更新到 Red Hat Developer Toolset 11.0。如果您之前已安装了 Red Hat Developer Toolset 的任何预发行版本，请从您的系统中卸载，如 第 1.7 节 "卸载 Red Hat Developer Toolset" 所述，安装新版本 第 1.5 节 "安装 Red Hat Developer Toolset"。

1.6.2. 更新至主版本

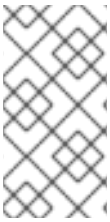
当有新的 Red Hat Developer Toolset 主版本时，您可以与之前的版本并行安装。有关如何在系统中安装 Red Hat Developer Toolset 的详情，请参考 第 1.5 节 "安装 Red Hat Developer Toolset"。

1.7. 卸载 RED HAT DEVELOPER TOOLSET

从系统中卸载 Red Hat Developer Toolset 软件包：

```
# yum remove devtoolset-11\* libasan libatomic libcilkrts libitm liblsan libtsan libubsan
```

这会从系统中删除 **GNU Compiler Collection**、**GNU Debugger**、**binutils** 以及系统所属的其他软件包。



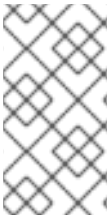
注意

Red Hat Enterprise Linux 7 的 Red Hat Developer Toolset 11.0 不再包括 **libatomic** 和 **libitm** 库，上面的命令会尝试删除，因为这个命令不需要在该系统上的正确功能 Red Hat Developer Toolset 组件的功能。然而，即使在 Red Hat Enterprise Linux 7 中，上述命令也会按预期工作。

请注意，卸载 Red Hat Developer Toolset 提供的工具不会影响这些工具的 Red Hat Enterprise Linux 系统版本。

1.8. 使用 RED HAT DEVELOPER TOOLSET 容器镜像

Docker 格式的容器镜像 可用于在虚拟软件容器内运行 Red Hat Developer Toolset 组件，从而将它们与主机系统隔离开来，并允许快速部署。有关 Red Hat Developer Toolset docker 格式的容器镜像和红帽开发人员工具集 Dockerfile 的详细信息，请参阅使用 Red Hat Software Collections 容器镜像 。



注意

docker 软件包包含 **Docker** 守护进程、命令行工具以及用于构建和使用 docker 格式的容器镜像的其他必要组件，目前仅适用于 Red Hat Enterprise Linux 7 产品的服务器变体。

按照 RHEL 7 中获取 Docker 的说明，设置供构建和使用 docker 格式的容器镜像的环境。

1.9. 其它资源

有关 Red Hat Developer Toolset 和 Red Hat Enterprise Linux 的详情，请查看下面列出的资源。

在线文档

- Red Hat Subscription Management 指南 - Red Hat Subscription Management collection 指南提供了如何管理 Red Hat Enterprise Linux 中的订阅的详细信息。
- Red Hat Developer Toolset 11.0 发行注记 - Red Hat Developer Toolset 11.0 发行注记。
- Red Hat Enterprise Linux 7 开发人员指南 - Red Hat Enterprise Linux 7 开发人员指南提供了有关 **Eclipse** IDE、库和运行时支持、编译和构建、调试和分析的更多信息。
- Red Hat Enterprise Linux 7 安装指南 - Red Hat Enterprise Linux 7 安装指南 解释了如何获取、安装和更新系统。
- Red Hat Enterprise Linux 7 系统管理员指南 - Red Hat Enterprise Linux 7 的 系统管理员指南 介绍了与 Red Hat Enterprise Linux 7 部署、配置和管理有关的信息。
- 使用 Red Hat Software Collections 容器镜像 - 本书提供了有关如何根据 Red Hat Software Collections 使用容器镜像的信息。可用的容器镜像包括应用程序、守护进程、数据库以及 Red Hat Developer Toolset 容器镜像。镜像可以在 Red Hat Enterprise Linux 7 服务器和 Red Hat Enterprise Linux Atomic Host 上运行。
- 容器入门 - 指南包含有关在 Red Hat Enterprise Linux 7 和 Red Hat Enterprise Linux Atomic Host 上构建和使用容器镜像的信息。

另请参阅

- *附录 A, 版本 11.0 的更改 - 与之前版本的 Red Hat Developer Toolset 工具版本相比，变更和增强列表。*

部分 II. 开发工具

第 2 章 GNU COMPILER COLLECTION(GCC)

GNU Compiler Collection （通常简写为 **GCC** ）是一个可移植的编译器套件，它支持广泛的编程语言。

Red Hat Developer Toolset 由 **GCC 11.2** 提供。此版本高于 Red Hat Enterprise Linux 中包含的版本，并提供了很多程序错误修复和增强。

2.1. GNU C COMPILER

2.1.1. 安装 C Compiler

在 Red Hat Developer Toolset 中，GNU C 编译器由 **devtoolset-11-gcc** 软件包提供，并使用 **devtoolset-11-toolchain** 自动安装，如 第 1.5 节“安装 Red Hat Developer Toolset”所述。

2.1.2. 使用 C Compiler

要在命令行中编译 C 程序，请运行 **gcc** 编译器，如下所示：

```
$ scl enable devtoolset-11 'gcc -o output_file source_file...'
```

这会在当前工作目录中创建名为 output_file 的二进制文件。如果省略 **-o** 选项，编译器会默认创建一个名为 **a.out** 的文件。

在处理由多个源文件组成的项目时，通常会先为每个源文件编译对象文件，然后将这些对象文件链接到一起。这样，当您更改单个源文件时，您只能重新编译此文件，而无需编译整个项目。要在命令行中编译对象文件，请执行以下操作：

```
$ scl enable devtoolset-11 'gcc -o object_file -c source_file'
```

这将创建一个名为 object_file 的对象文件。如果省略 **-o** 选项，编译器会创建一个名为 的文件，该文件名为 **.o** 文件扩展名。将对象文件链接到一起并创建二进制文件：

```
$ scl enable devtoolset-11 'gcc -o output_file object_file...'
```

请注意，您可以使用 **scl** 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset **gcc** 作为默认值运行 shell 会话：

```
$ scl enable devtoolset-11 'bash'
```

注意

验证您使用 **gcc** 的版本：

```
$ which gcc
```

Red Hat Developer Toolset 的 **gcc** 可执行路径以 **/opt** 开头。另外，您可以使用以下命令确认与 Red Hat Developer Toolset **gcc** 的版本号匹配：

```
$ gcc -v
```

例 2.1. 在命令行中编译 C 程序

考虑名为 **hello.c** 的源文件，其内容如下：

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello, World!\n");
  return 0;
}
```

使用 Red Hat Developer Toolset 的 **gcc** 编译器在命令行中编译此源代码：

```
$ scl enable devtoolset-11 'gcc -o hello hello.c'
```

这会在当前工作目录中创建一个名为 **hello** 的新二进制文件。

2.1.3. 运行 C 程序

当 **gcc** 编译程序时，它会创建一个可执行的二进制文件。要在命令行中运行这个程序，请切换到包含可执行文件的目录并运行它：

```
$ ./file_name
```

例 2.2. 在命令行中运行 C 程序

假设您已像 例 2.1 “在命令行中编译 C 程序” 所示成功编译了 **hello** 二进制文件，您可以在 shell 提示符后输入以下内容来运行该文件：

```
$ ./hello
Hello, World!
```

2.2. GNU C++ 编译器

2.2.1. 安装 C++ Compiler

在 Red Hat Developer Toolset 中，GNU C++ 编译器由 **devtoolset-11-gcc-c++** 软件包提供，并会自动使用 **devtoolset-11-toolchain** 软件包安装，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

2.2.2. 使用 C++ Compiler

要在命令行中编译 C++ 程序，请运行 **g++** 编译器，如下所示：

```
$ scl enable devtoolset-11 'g++ -o output_file source_file...'
```

这会在当前工作目录中创建名为 output_file 的二进制文件。如果省略 **-o** 选项，则 **g++** 编译器会默认创建一个名为 **a.out** 的文件。

在处理由多个源文件组成的项目时，通常会先为每个源文件编译对象文件，然后将这些对象文件链接到一起。这样，当您更改单个源文件时，您只能重新编译此文件，而无需编译整个项目。在命令行中编译对象文件：

```
$ scl enable devtoolset-11 'g++ -o object_file -c source_file'
```

这将创建一个名为 object_file 的对象文件。如果省略 **-o** 选项，则 **g++** 编译器会在源文件后创建一个名为 **.o** 文件扩展名的文件。将对象文件链接到一起并创建二进制文件：

```
$ scl enable devtoolset-11 'g++ -o output_file object_file...'
```

请注意，您可以使用 **scl** 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset **g++** 作为默认值运行 shell 会话：

```
$ scl enable devtoolset-11 'bash'
```

注意

在任意时间点上验证您使用的 **g++** 版本：

```
$ which g++
```

Red Hat Developer Toolset 的 **g++** 可执行路径以 /opt 开头。另外，您可以使用以下命令确认与 Red Hat Developer Toolset **g++** 匹配的版本号：

```
$ g++ -v
```

例 2.3. 在命令行上编译 C++ 程序

考虑名为 **hello.cpp** 的源文件，其内容如下：

```
#include <iostream>

using namespace std;

int main(int argc, char *argv[]) {
  cout << "Hello, World!" << endl;
  return 0;
}
```

使用 Red Hat Developer Toolset 中的 **g++** 编译器在命令行中编译此源代码：

```
$ scl enable devtoolset-11 'g++ -o hello hello.cpp'
```

这会在当前工作目录中创建一个名为 **hello** 的新二进制文件。

2.2.3. 运行 C++ 计划

当 **g++** 编译程序时，它会创建一个可执行的二进制文件。要在命令行中运行这个程序，请切换到包含可执行文件的目录并运行它：

```
$ ./file_name
```

例 2.4. 在命令行中运行 C++ 程序

假设您已成功编译了 **hello** 二进制文件，如 例 2.3 "在命令行上编译 C++ 程序" 所示，您可以运行它：

```
$ ./hello
Hello, World!
```

2.2.4. c++ 兼容性

所有 Red Hat Enterprise Linux 版本 5、6 和 7 以及 Red Hat Developer Toolset 版本 1 到 10 的所有编译器均与 C++98 模式中的任何其他编译器兼容。

C++11、C++14 或 C++17 模式中的编译器只能保证在同一模式下与另一编译器兼容。

支持的示例：

- **Red Hat Developer Toolset 6.x 中的 C++11 和 C++11**
- **Red Hat Developer Toolset 6.x 中的 C++14 和 C++14**
- **来自 Red Hat Developer Toolset 10.x 的 C++17 和 C++17**

重要

- **Red Hat Developer Toolset 10.x 中的 GCC 编译器可以使用 C++20 构建代码，但这个功能是实验性的，且不受红帽支持。**
- **本节所述的所有兼容性信息都仅适用于 GCC C++ 编译器的红帽版本。**

2.2.4.1. C++ ABI

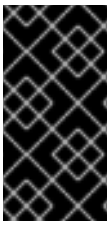
使用 -std=c++98 或 -std=gnu++98 或 -std=gnu++98 可自由混合使用 Red Hat Enterprise Linux

5、6 或 7 系统 GCC 构建的、与 Red Hat Developer Toolset 工具链构建的 C++98 的二进制文件或库。

Red Hat Developer Toolset 11.0 的默认语言标准设置是 C++17，具有 GNU 扩展，相当于使用选项 -std=gnu++17。

当使用 Red Hat Developer Toolset 6 或更高版本构建时，Red Hat Developer Toolset 中支持使用 C++14 语言版本，所有使用相应标志编译的 C++ 对象。C++98 中由系统 GCC 编译的对象也兼容，但在 C++11 或 C++14 模式中，使用系统 GCC 编译的对象不兼容。

从 Red Hat Developer Toolset 10.x 开始，使用 C++17 语言版本不再是实验性的，由红帽提供支持。所有使用 C++17 编译的 C++ 对象必须使用 Red Hat Developer Toolset 10.x 或更高版本构建。



重要

使用应用程序中的 C++11、C++14 和 C++17 功能需要仔细考虑上述 ABI 兼容性信息。

使用 Red Hat Developer Toolset 中的 GCC 明确支持由 Red Hat Enterprise Linux 7 系统工具 chain GCC 构建的对象、二进制文件和库的组合。

除了上面讨论的 C++11、C++14 和 C++17 ABI 外，Red Hat Enterprise Linux 应用程序兼容性规范不会改变用于 Red Hat Developer Toolset。当使用通过 Red Hat Enterprise Linux 7 工具链（一个 .o/.a 文件）构建的 Red Hat Developer Toolset 工具链结合使用时，Red Hat Developer Toolset 工具链应用于任何链接。这样可确保只由 Red Hat Developer Toolset 提供的更新库功能会在链接时解析。

添加了一个新的标准控制 SIMD vector 类型，以避免在有向量不同长度的系统中进行名称冲突。Red Hat Developer Toolset 中的编译器默认使用新的 mangling。通过在 GCC C++ 编译器调用中添加 -fabi-version=2 或 -fabi-version=3 选项，可以使用前面的标准 mangling。要显示使用旧 mangling 的代码的警告，请使用 -Wabi 选项。

在 Red Hat Enterprise Linux 7 中，GCC C++ 编译器默认仍然使用旧的 mangling，但会在支持强大别名的目标上发送别名。通过将 -fabi-version=4 选项添加到编译器调用，可以使用新的标准 mangling。要显示使用旧 mangling 的代码的警告，请使用 -Wabi 选项。

在 Red Hat Enterprise Linux 7 中，Red Hat Developer Toolset 中的 GCC C++ 编译器仍然使用 std::string 的旧参考计数实现。这是为了与 Red Hat Enterprise Linux 7 系统工具chain GCC 的兼容

性。这意味着，即使使用 Red Hat Developer Toolset 编译器也是如此，即使使用 Red Hat Developer Toolset 编译器也会提供一些新的 C++17 功能，如 std::pmr::string 也是如此。

2.3. GNU FORTRAN COMPILER

2.3.1. 安装 Fortran Compiler

在 Red Hat Developer Toolset 中，GNU Fortran 编译器由 devtoolset-11-gcc-gfortran 软件包提供，并会自动安装 devtoolset-11-toolchain，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

2.3.2. 使用 Fortran Compiler

要在命令行中编译 Fortran 程序，请运行 gfortran 编译器，如下所示：

```
$ scl enable devtoolset-11 'gfortran -o output_file source_file...'
```

这会在当前工作目录中创建名为 output_file 的二进制文件。如果省略 -o 选项，编译器会默认创建一个名为 a.out 的文件。

在处理由多个源文件组成的项目时，通常会先为每个源文件编译对象文件，然后将这些对象文件链接到一起。这样，当您更改单个源文件时，您只能重新编译此文件，而无需编译整个项目。在命令行中编译对象文件：

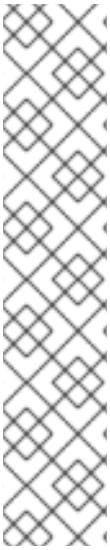
```
$ scl enable devtoolset-11 'gfortran -o object_file -c source_file'
```

这将创建一个名为 object_file 的对象文件。如果省略 -o 选项，编译器会创建一个名为 的文件，该文件名为 .o 文件扩展名。将对象文件链接到一起并创建二进制文件：

```
$ scl enable devtoolset-11 'gfortran -o output_file object_file...'
```

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset gfortran 运行 shell 会话作为默认值：

```
$ scl enable devtoolset-11 'bash'
```



注意

验证您使用 gfortran 的版本：

```
$ which gfortran
```

Red Hat Developer Toolset 的 gfortran 可执行路径从 /opt 开始。另外，您可以使用以下命令确认与 Red Hat Developer Toolset gfortran 匹配的版本号：

```
$ gfortran -v
```

例 2.5. 在命令行上编译 Fortran 程序

考虑名为 hello.f 的源文件，其内容如下：

```
program hello
  print *, "Hello, World!"
end program hello
```

使用 Red Hat Developer Toolset 的 gfortran 编译器在命令行中编译此源代码：

```
$ scl enable devtoolset-11 'gfortran -o hello hello.f'
```

这会在当前工作目录中创建一个名为 hello 的新二进制文件。

2.3.3. 运行 Fortran 计划

当 gfortran 编译程序时，它会创建一个可执行的二进制文件。要在命令行中运行这个程序，请切换到包含可执行文件的目录并运行它：

```
$ ./file_name
```

例 2.6. 在命令行上运行 Fortran Program

假设您已成功编译了 hello 二进制文件，如 *例 2.5 “在命令行上编译 Fortran 程序”* 所示，您可以运行它：

```
$ ./hello
 Hello, World!
```

2.4. RED HAT DEVELOPER TOOLSET 中的 GCC 细节

库的静态链接

某些最新库功能会静态链接到使用 Red Hat Developer Toolset 构建的应用程序中，以支持对多个版本的 Red Hat Enterprise Linux 执行。这会造成额外的微小的安全风险，因为标准 Red Hat Enterprise Linux 勘误不会改变这个代码。如果开发人员由于此类风险而重建其应用程序的需求，红帽将使用安全勘误进行沟通。

重要

由于这种额外的安全风险，强烈建议开发人员不以静态方式将其整个应用程序链接到同一原因。

在链接对象文件后指定库

在 Red Hat Developer Toolset 中，库使用链路器脚本链接，这些脚本可能会通过静态归档指定某些符号。这需要确保与多个 Red Hat Enterprise Linux 版本兼容。但是，linker 脚本使用对应共享对象文件的名称。因此，链接者使用不同于预期的符号处理规则，当选项在指定对象文件前指定库时，不会识别对象文件所需的符号：

```
$ scl enable devtoolset-11 'gcc -lsomelib objfile.o'
```

以这种方式使用 Red Hat Developer Toolset 的库会导致链接器错误消息 undefined 引用到符号。要防止这个问题，请按照标准链接实践操作，并在指定对象文件选项后指定添加库的选项：

```
$ scl enable devtoolset-11 'gcc objfile.o -lsomelib'
```

请注意，在使用 GCC 基本 Red Hat Enterprise Linux 版本时，这个建议也会适用。

2.5. 其它资源

有关 GNU Compiler Collections 及其功能的更多信息，请参阅以下列出的资源。

安装的文档

-

GCC(1)- gcc 编译器的 man page 提供了有关其用途的详细信息，而 g++ 则接受与 gcc 相同的命令行选项。显示 Red Hat Developer Toolset 中包含的版本的 man page：

```
$ scl enable devtoolset-11 'man gcc'
```

- gfortran(1)- gfortran 编译器的 man page 提供其用法的详细信息。显示 Red Hat Developer Toolset 中包含的版本的 man page：

```
$ scl enable devtoolset-11 'man gfortran'
```

- C++ 标准库文档 - C++ 标准库中的文档可以选择性地安装：

```
# yum install devtoolset-11-libstdc++-docs
```

安装后，HTML 文档位于 /opt/rh/devtoolset-11/root/usr/share/doc/devtoolset-11-libstdC++-docs-11.2/html/index.html。

在线文档

- Red Hat Enterprise Linux 7 开发人员指南 - Red Hat Enterprise Linux 7 开发人员指南 提供了有关 GCC 的深入信息。
- 使用 GNU Compiler Collection - 上游 GCC 手册提供 GNU 编译器及其用法的深入描述。
- GNU C++ 库 - GNU C++ 库文档提供有关标准 C++ 库的 GNU 实施的详细信息。
- GNU Fortran Compiler - GNU Fortran 编译器文档提供了有关 gfortran 的用法的详细信息。

另请参阅

- 第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。
- 第 4 章 binutils - 使用 binutils 的说明，这是用于检查和操作对象文件和二进制文件的二进制工具集合。

- *第 5 章 elfutils - 使用 elfutils 的说明，这是检查和操作 ELF 文件的二进制工具集合。*

- *第 6 章 dwz - 使用 dwz 工具优化 ELF 共享库和 ELF 可执行文件大小中包含的 DWARF 调试信息的说明。*

- *第 8 章 GNU 调试器(GDB) - 关于在 C、C++ 和 Fortran 中编写的调试程序的说明。*

第 3 章 GNU MAKE

GNU make 实用程序通常缩写为生成来自源文件的可执行文件的工具。使 自动编译复杂程序中的哪些部分已经更改，需要重新编译。make 使用名为 Makefile 的配置文件来控制程序的构建方式。

Red Hat Developer Toolset 带有 make 4.3。此版本高于 Red Hat Enterprise Linux 中包含的版本，并提供了很多程序错误修复和增强。

3.1. 安装 MAKE

在 Red Hat Developer Toolset 中，GNU make 由 devtoolset-11-make 软件包提供，并自动安装 devtoolset-11-toolchain，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

3.2. 使用 MAKE

要在不使用 Makefile 的情况下构建程序，请运行 make 工具，如下所示：

```
$ scl enable devtoolset-11 'make source_file_without_extension'
```

此命令利用为多种编程语言（包括 C、C++ 和 Fortran）定义隐式规则。其结果是当前工作目录中名为 source_file_without_extension 的二进制文件。

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset make 运行 shell 会话作为默认设置：

```
$ scl enable devtoolset-11 'bash'
```

注意

在任意时间点验证您使用的版本：

```
$ which make
```

Red Hat Developer Toolset 的 make 可执行路径从 /opt 开始。另外，您可以使用以下命令确认与 Red Hat Developer Toolset 的版本号匹配：

```
$ make -v
```

例 3.1. 使用 make 构建 C 计划

考虑名为 hello.c 的源文件，其内容如下：

```
#include <stdio.h>

int main(int argc, char *argv[]) {
  printf("Hello, World!\n");
  return 0;
}
```

使用 Red Hat Developer Toolset 中 make 工具定义的隐式规则构建此源代码：

```
$ scl enable devtoolset-11 'make hello'
cc     hello.c   -o hello
```

这会在当前工作目录中创建一个名为 hello 的新二进制文件。

3.3. 使用 MAKEFILE

要构建由多个源文件组成的复杂程序，使用名为 Makefile 的配置文件，该文件控制如何编译程序组件并构建最终可执行文件。 Makefile 也可以包含清理工作目录、安装和升级程序文件以及其它操作的说明。

make 自动使用当前目录中名为 GNUmakefile、makefile 或 Makefile 的文件。要指定其他文件名，使用 -f 选项：

```
$ make -f make_file
```

描述 Makefile 语法的详细信息已超出本指南的范围。请参阅 GNU make （上游 GNU make manual）提供了 GNU make 实用程序、Makefile 语法及其用法的深入描述。

完整的 make manual 也以 Texinfo 格式作为安装的一部分提供。查看这个手册：

```
$ scl enable devtoolset-11 'info make'
```

例 3.2. 使用 Makefile 构建 C 程序

考虑以下名为 Makefile 的 Makefile 来构建 例 3.1 “使用 make 构建 C 计划” 中引入的简单 C 程序。Makefile 定义一些变量并指定四个 规则，它们由 目标 及其 配方 组成。请注意，带有方法的行必须以 TAB 字符开头：

```
CC=gcc
CFLAGS=-c -Wall
SOURCE=hello.c
OBJ=$(SOURCE:.c=.o)
EXE=hello

all: $(SOURCE) $(EXE)

$(EXE): $(OBJ)
	$(CC) $(OBJ) -o $@

.o: .c
	$(CC) $(CFLAGS) $< -o $@

clean:
	rm -rf $(OBJ) $(EXE)
```

要使用这个 Makefile 构建 hello.c 程序，请运行 make 工具程序：

```
$ scl enable devtoolset-11 'make'
gcc -c -Wall hello.c -o hello.o
gcc hello.o -o hello
```

这会在当前工作目录中创建一个新的对象文件 hello.o 和名为 hello 的新二进制文件。

要清理工作目录，请运行：

```
$ scl enable devtoolset-11 'make clean'
rm -rf hello.o hello
```

这会从工作目录中删除对象和二进制文件。

3.4. 其它资源

有关 GNU make 工具及其功能的更多信息，请参阅以下列出的资源。

安装的文档

- make(1)- make 实用程序的 man page 提供其用法信息。显示 Red Hat Developer Toolset 中包含的版本的 man page：

  ```
  $ scl enable devtoolset-11 'man make'
  ```

- 完整的 make manual（包含 Makefile 语法的详细信息）也以 Texinfo 格式提供。显示 Red Hat Developer Toolset 中包含的版本的信息手册：

  ```
  $ scl enable devtoolset-11 'info make'
  ```

在线文档

- GNU make - 上游 GNU make 手册提供了 GNU make 实用程序、Makefile 语法及其用法的深入描述。

另请参阅

- 第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。
- 第 2 章 GNU Compiler Collection(GCC) - 使用 GNU Compiler Collection 的说明，这是可移植编译器套件，支持广泛的编程语言。
- 第 4 章 binutils - 使用 binutils 的说明，这是用于检查和操作对象文件和二进制文件的二进制工具集合。

- *第 5 章 elfutils - 使用 elfutils 的说明，这是检查和操作 ELF 文件的二进制工具集合。*

- *第 6 章 dwz - 使用 dwz 工具优化 ELF 共享库和 ELF 可执行文件大小中包含的 DWARF 调试信息的说明。*

- *第 8 章 GNU 调试器(GDB) - 关于在 C、C++ 和 Fortran 中编写的调试程序的说明。*

第 4 章 BINUTILS

binutils 是各种二进制工具的集合，如 GNU linker、GNU 汇编器 以及其他实用程序，允许您检查和操作对象文件和二进制文件。如需了解与 Red Hat Developer Toolset 版本 binutils 分发的二进制工具的完整列表，请参阅 表 4.1 “binutils 中包括用于 Red Hat Developer Toolset 的工具”。

Red Hat Developer Toolset 带有 binutils 2.36。此版本比 Red Hat Enterprise Linux 及以前的 Red Hat Developer Toolset 版本中包含的版本提供程序错误修正和增强。

表 4.1. binutils 中包括用于 Red Hat Developer Toolset 的工具

Name	描述
addr2line	将地址转换为文件名和行号。
ar	从存档创建、修改和提取文件。
as	GNU 汇编器.
c++filt	解码 mangled C++ 符号。
dwp	将 DWARF 对象文件合并到一个 DWARF 软件包文件中。
elfedit	检查和编辑 ELF 文件。
gprof	显示性能分析信息。
ld	GNU 链路器.
ld.bfd	GNU 链路器的替代选择。
ld.gold	GNU 链路器的另一个选择。
nm	列出对象文件的符号。
objcopy	复制和转换对象文件。
objdump	显示对象文件的信息。
ranlib	生成存档内容的索引，以便更快地访问此存档。
readelf	显示有关 ELF 文件的信息。
size	列出对象或存档文件的部分大小。

Name	描述
字符串	显示文件中可打印的字符序列。
strip	忽略对象文件中的所有符号。

4.1. 安装 BINUTILS

在 Red Hat Developer Toolset 中，binutils 由 devtoolset-11-binutils 软件包提供，并会自动安装 devtoolset-11-toolchain，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

4.2. 使用 GNU ASSEMBLER

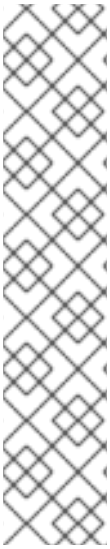
要从装配语言程序生成对象文件，请使用 如下 工具运行：

```
$ scl enable devtoolset-11 'as option ... -o object_file source_file'
```

这会在当前工作目录中创建名为 object_file 的对象文件。

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset 作为默认 运行 shell 会话：

```
$ scl enable devtoolset-11 'bash'
```



注意

在任意时间点上验证您使用的版本：

```
$ which as
```

Red Hat Developer Toolset 的可执行路径 以 /opt 开始。另外，您可以使用以下命令确认 与 Red Hat Developer Toolset 的版本号匹配：

```
$ as -v
```

4.3. 使用 GNU LINKER

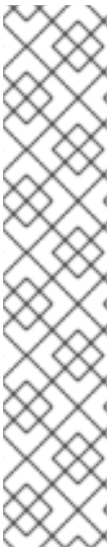
要从对象文件创建可执行文件或库，请按如下方式运行 ld 工具：

```
$ scl enable devtoolset-11 'ld option ... -o output_file object_file ...'
```

这会在当前工作目录中创建名为 output_file 的二进制文件。如果省略 -o 选项，编译器会默认创建一个名为 a.out 的文件。

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您以 Red Hat Developer Toolset ld 运行 shell 会话：

```
$ scl enable devtoolset-11 'bash'
```



注意

在任意时间点验证您使用的版本：

```
$ which ld
```

Red Hat Developer Toolset 的 ld 可执行路径以 /opt 开始。另外，您可以使用以下命令确认与 Red Hat Developer Toolset ld 的版本号匹配：

```
$ ld -v
```

4.4. 使用其他二进制工具

binutils 提供除 linker 和 assembler 以外的许多二进制工具。有关这些工具的完整列表，请参阅 表 4.1 “binutils 中包括用于 Red Hat Developer Toolset 的工具”。

执行属于 binutils 一部分的任何工具：

```
$ scl enable devtoolset-11 'tool option ... file_name'
```

如需通过 binutils 分发的工具列表，请参阅 表 4.1 “binutils 中包括用于 Red Hat Developer Toolset 的工具”。例如，使用 objdump 工具检查对象文件：

```
$ scl enable devtoolset-11 'objdump option ... object_file'
```

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset 二进制工具运行 shell 会话，默认：

```
$ scl enable devtoolset-11 'bash'
```

注意

在任意时间点上验证您使用的 binutils 版本：

```
$ which objdump
```

Red Hat Developer Toolset 的 objdump 可执行路径从 /opt 开始。另外，您可以使用以下命令确认与 Red Hat Developer Toolset objdump 的版本号匹配：

```
$ objdump -v
```

4.5. RED HAT DEVELOPER TOOLSET 中的 BINUTILS 的具体内容

库的静态链接

某些最新库功能会静态链接到使用 Red Hat Developer Toolset 构建的应用程序中，以支持对多个版本的 Red Hat Enterprise Linux 执行。这会造成额外的微小的安全风险，因为标准 Red Hat Enterprise Linux 勘误不会改变这个代码。如果开发人员由于此类风险而重建其应用程序的需求，红帽将使用安全勘误进行沟通。

重要

由于这种额外的安全风险，强烈建议开发人员不以静态方式将其整个应用程序链接到同一原因。

在链接对象文件后指定库

在 Red Hat Developer Toolset 中，库使用链路器脚本链接，这些脚本可能会通过静态归档指定某些符号。这需要确保与多个 Red Hat Enterprise Linux 版本兼容。但是，linker 脚本使用对应共享对象文件的名称。因此，链接者使用不同于预期的符号处理规则，当选项在指定对象文件前指定库时，不会识别对象文件所需的符号：

```
$ scl enable devtoolset-11 'ld -lsomelib objfile.o'
```

以这种方式使用 Red Hat Developer Toolset 的库会导致链接器错误消息 undefined 引用到符号。要防止这个问题，请按照标准链接实践操作，并在指定对象文件选项后指定添加库的选项：

```
$ scl enable devtoolset-11 'ld objfile.o -lsomelib'
```

请注意，在使用基本 Red Hat Enterprise Linux 版本 binutils 时，这个建议也会适用。

4.6. 其它资源

有关 binutils 的更多信息，请查看下面列出的资源。

安装的文档

- 对于(1), ld(1), addr2line(1), ar(1), c++filt(1), dwp(1), elfedit(1), gprof(1), nm(1), objcopy(1), objdump(1), runlib(1), readelf(1), size(1), 字符串(1), strip(1), - Manual page for various binutils 工具提供了有关各自用途的更多信息。显示 Red Hat Developer Toolset 中包含的版本的 man page：

  ```
  $ scl enable devtoolset-11 'man tool'
  ```

在线文档

- binutils - binutils 文档提供了对二进制工具及其用法的深入描述。

另请参阅

- 第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。
- 第 5 章 elfutils - 有关如何使用 elfutils、用于检查和操作 ELF 文件的二进制工具集合的信息。
- 第 2 章 GNU Compiler Collection(GCC) - 如何编译在 C、C++ 和 Fortran 中编写的程序的信息。

第 5 章 ELFUTILS

elfutils 是各种二进制工具的集合，如 eu-objdump、eu-readelf 以及其他可让您检查和操作 ELF 文件的工具。如需了解使用 Red Hat Developer Toolset 版本发布的完整二进制工具列表，请参阅 表 5.1 “elfutils for Red Hat Developer Toolset 中包含的工具”。

Red Hat Developer Toolset 带有 elfutils 0.185。此版本比 Red Hat Developer Toolset 的以前版本包括了一个版本，它提供了一些程序错误修复和增强。

表 5.1. elfutils for Red Hat Developer Toolset 中包含的工具

Name	描述
eu-addr2line	将地址转换为文件名和行号。
eu-ar	从存档创建、修改和提取文件。
eu-elfcmp	比较两个 ELF 文件中的相关部分相等。
eu-elflint	验证 ELF 文件是否符合 *通用 ABI*(**gABI**)和 *特定于处理器的补充 ABI* (**psABI**)规格。
eu-findtextrel	在文件中找到文本重新定位的源。
eu-make-debug-archive	创建离线存档以用于调试。
eu-nm	列出对象文件的符号。
eu-objdump	显示对象文件的信息。
eu-ranlib	生成存档内容的索引，以便更快地访问此存档。
eu-readelf	显示有关 ELF 文件的信息。
eu-size	列出对象或存档文件的部分大小。
eu-stack	用于取消处理进程和核心的新实用程序。
eu-strings	显示文件中可打印的字符序列。
eu-strip	忽略对象文件中的所有符号。
eu-unstrip	组合了带有独立符号和调试信息的条状化文件。

5.1. 安装 ELFUTILS

在 Red Hat Developer Toolset 中，devtoolset-11-elfutils 软件包提供 elfutils，它会自动安装 devtoolset-11-toolchain，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

5.2. 使用 ELFUTILS

要执行属于 elfutils 的任何工具，请按如下所示运行该工具：

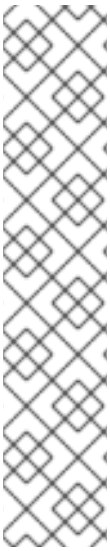
```
$ scl enable devtoolset-11 'tool option ... file_name'
```

如需 elfutils 分发的工具列表，请参阅 表 5.1 “elfutils for Red Hat Developer Toolset 中包含的工具”。例如，使用 eu-objdump 工具检查对象文件：

```
$ scl enable devtoolset-11 'eu-objdump option ... object_file'
```

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset 二进制工具运行 shell 会话，默认：

```
$ scl enable devtoolset-11 'bash'
```



注意

要验证 elfutils 的版本，您要在任何时间点上使用：

```
$ which eu-objdump
```

Red Hat Developer Toolset 的 eu-objdump 可执行路径从 /opt 开始。另外，您可以使用以下命令确认与 Red Hat Developer Toolset eu-objdump 的版本号匹配：

```
$ eu-objdump -V
```

5.3. 其它资源

有关 elfutils 的更多信息，请查看下面列出的资源。

另请参阅

-

第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。

- *第 2 章 GNU Compiler Collection(GCC) - 在编译程序上以 C、C++ 和 Fortran 编写的说明。*
- *第 4 章 binutils - 使用 binutils 的说明，这是用于检查和操作对象文件和二进制文件的二进制工具集合。*
- *第 6 章 dwz - 使用 dwz 工具优化 ELF 共享库和 ELF 可执行文件大小中包含的 DWARF 调试信息的说明。*

第 6 章 DWZ

dwz 是一个命令行工具，它会尝试优化 ELF 共享库中包含的 DWARF 调试信息以及 ELF 可执行文件的大小。为此，dwz 将 DWARF 信息表示为可能，使用 DWARF 标准 附录 E 中的技术降低重复数量。

Red Hat Developer Toolset 带有 dwz 0.14。

6.1. 安装 DWZ

在 Red Hat Developer Toolset 中，devtoolset-11- dwz 软件包提供的 dwz 工具，它会自动安装为 devtoolset-11-toolchain，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

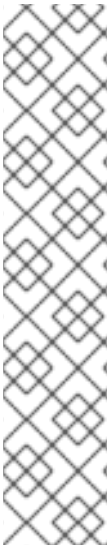
6.2. 使用 DWZ

要在二进制文件中优化 DWARF 调试信息，请按如下方式运行 dwz 工具：

```
$ scl enable devtoolset-11 'dwz option... file_name'
```

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset dwz 作为默认值运行 shell 会话：

```
$ scl enable devtoolset-11 'bash'
```



注意

验证您要使用的 dwz 版本：

```
$ which dwz
```

Red Hat Developer Toolset 的 dwz 可执行路径以 /opt 开头。另外，您可以使用以下命令确认与 Red Hat Developer Toolset dwz 匹配的版本号：

```
$ dwz -v
```

6.3. 其它资源

有关 dwz 及其功能的更多信息，请参阅以下列出的资源。

安装的文档

- dwz(1)- dwz 实用程序的 man page 提供其用法的详细信息。显示 Red Hat Developer Toolset 中包含的版本的 man page：

```
$ scl enable devtoolset-11 'man dwz'
```

另请参阅

- 第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。
- 第 2 章 GNU Compiler Collection(GCC) - 在编译程序上以 C、C++ 和 Fortran 编写的说明。
- 第 4 章 binutils - 使用 binutils 的说明，这是用于检查和操作对象文件和二进制文件的二进制工具集合。
- 第 5 章 elfutils - 使用 elfutils 的说明，这是检查和操作 ELF 文件的二进制工具集合。

第 7 章 ANNOBIN

Annobin 项目由 annobin 插件和 annockeck 程序组成。

anobin 插件扫描 GNU Compiler Collection(GCC)命令行、编译状态和编译过程，并生成 ELF 备注。ELF 备注记录了二进制文件的构建方式，并为 annocheck 程序提供信息来执行安全强化检查。

安全强化检查程序是 annocheck 程序的一部分，默认是启用的。它检查二进制文件，以确定程序是否使用必要的安全强化选项构建并正确编译。annocheck 可以递归扫描 ELF 对象文件的目录、存档和 RPM 软件包。

注意

这些文件必须采用 ELF 格式。annocheck 不处理任何其他二进制文件类型。

7.1. 安装 ANNOBIN

在 Red Hat Developer Toolset 中，annobin 插件和 annockeck 程序由 devtoolset-11-gcc 软件包提供，并如 第 1.5.3 节 “安装可选软件包” 所述。

7.2. 使用 ANNOBIN 插件

要将选项传递给带有 gcc 的 annobin 插件，请使用：

```
$ scl enable devtoolset-11 'gcc -fplugin=annobin -fplugin-arg-annobin-option file-name'
```

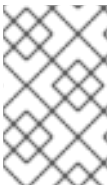
请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset 作为默认 运行 shell 会话：

```
$ scl enable devtoolset-11 'bash'
```

7.3. 使用 ANNOCHECK

使用 annocheck 程序扫描文件、目录或 RPM 软件包：

```
$ scl enable devtoolset-11 'annocheck file-name'
```

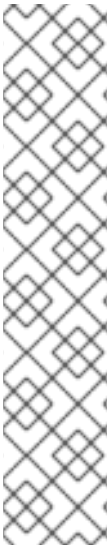


注意

annocheck 仅查找 ELF 文件。其他文件类型将被忽略。

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset 作为默认 运行 shell 会话：

```
$ scl enable devtoolset-11 'bash'
```



注意

验证您使用的 annocheck 版本：

```
$ which annocheck
```

Red Hat Developer Toolset 的 annocheck 可执行路径从 /opt 开始。另外，您可以使用以下命令确认与 Red Hat Developer Toolset annocheck 匹配的版本号：

```
$ annocheck --version
```

7.4. 其它资源

有关 annocheck、nobin 及其功能的更多信息，请参阅以下列出的资源。

安装的文档

- annocheck(1)- annocheck 工具的手册页提供了其用法的详细信息。显示 Red Hat Developer Toolset 中包含的版本的 man page：

  ```
  $ scl enable devtoolset-11 'man annocheck'
  ```

- annobin(1)- annobin 实用程序的 man page 提供了有关其使用情况的详细信息。显示 Red Hat Developer Toolset 中包含的版本的 man page：

```
$ scl enable devtoolset-11 'man annobin'
```

部分 III. 调试工具

第 8 章 GNU 调试器(GDB)

GNU Debugger （通常缩写为 GDB ）是一个命令行工具，可用于调试使用各种编程语言编写的程序。它允许您在被调试的代码中检查内存，控制代码的执行状态，检测代码的特定部分的执行，等等。

Red Hat Developer Toolset 带有 GDB 10.2。此版本比 Red Hat Enterprise Linux 以及之前发布的 Red Hat Developer Toolset 版本提供一些改进和多个程序错误修复。

8.1. 安装 GNU DEBUGGER

在 Red Hat Developer Toolset 中，GNU Debugger 由 devtoolset-11-gdb 软件包提供，并自动安装 devtoolset-11-toolchain，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

8.2. 准备进行调试的程序

使用调试信息编译程序

要使用 GNU Debugger 读取的调试信息编译 C 程序，请确保使用 -g 选项运行 gcc 编译器：

```
$ scl enable devtoolset-11 'gcc -g -o output_file input_file...'
```

同样，使用调试信息编译 C++ 程序：

```
$ scl enable devtoolset-11 'g++ -g -o output_file input_file...'
```

例 8.1. 使用调试信息编译 C 程序

考虑名为 fibonacci.c 的源文件，其内容如下：

```
#include <stdio.h>
#include <limits.h>

int main (int argc, char *argv[]) {
  unsigned long int a = 0;
  unsigned long int b = 1;
  unsigned long int sum;

  while (b < LONG_MAX) {
    printf("%ld ", b);
    sum = a + b;
    a = b;
    b = sum;
```

```
    }

    return 0;
}
```

使用 Red Hat Developer Toolset 中的 GCC 使用 GNU Debugger 的调试信息，在命令行中编译该程序：

```
$ scl enable devtoolset-11 'gcc -g -o fibonacci fibonacci.c'
```

这会在当前工作目录中创建一个名为 fibonacci 的新二进制文件。

安装现有软件包的调试信息

要为系统中已安装的软件包安装调试信息：

```
# debuginfo-install package_name
```

请注意，必须安装 yum-utils 软件包，以便您的系统中有 debuginfo-install 工具。

例 8.2. 为 glibc 软件包安装调试信息

安装 glibc 软件包的调试信息：

```
# debuginfo-install glibc
Loaded plugins: product-id, refresh-packagekit, subscription-manager
--> Running transaction check
---> Package glibc-debuginfo.x86_64 0:2.17-105.el7 will be installed
...
```

8.3. 运行 GNU DEBUGGER

要在您要调试的程序上运行 GNU Debugger ：

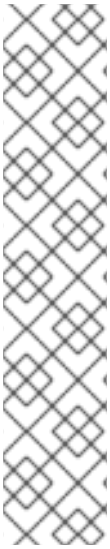
```
$ scl enable devtoolset-11 'gdb file_name'
```

这将以交互模式启动 gdb 调试器并显示默认提示符 (gdb)。要退出调试会话并返回到 shell 提示符，请随时运行以下命令：

```
(gdb) quit
```

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset gdb 作为默认值运行 shell 会话：

```
$ scl enable devtoolset-11 'bash'
```



注意

在任意时间点验证您使用的 gdb 版本：

```
$ which gdb
```

Red Hat Developer Toolset 的 gdb 可执行文件路径以 /opt 开头。另外，您可以使用以下命令确认与 Red Hat Developer Toolset gdb 的版本号匹配：

```
$ gdb -v
```

例 8.3. 在 fibonacci Binary 文件中运行 gdb 工具程序

这个示例假设您已成功编译了 fibonacci 二进制文件，如 例 8.1 “使用调试信息编译 C 程序” 所示。

使用 gdb 启动调试 fibonacci ：

```
$ scl enable devtoolset-11 'gdb fibonacci'
GNU gdb (GDB) Red Hat Enterprise Linux 8.2-2.el7
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from fibonacci...done.
(gdb)
```

```
-
```

8.4. 列出源代码

查看您要调试的程序的源代码：

```
(gdb) list
```

在开始执行要调试的程序前，g db 会显示源代码的前十个行，以及这个命令的后续使用列出了另外 10 行。开始执行后，g db 会显示周围执行所在的行，通常是在您设置断点时。

您还可以显示括起特定行的代码：

```
(gdb) list file_name:line_number
```

同样，要显示周围特定功能开头的代码：

```
(gdb) list file_name:function_name
```

请注意，您可以更改 list 命令显示的行数：

```
(gdb) set listsize number
```

例 8.4. 列出 fibonacci Binary 文件的源代码

在 例 8.1 “使用调试信息编译 C 程序” 中列出的 fibonacci.c 文件准确有 17 行。假设您已使用调试信息并希望 gdb 实用程序能够列出整个源代码，您可以运行以下命令来将列出的行数更改为 20：

```
(gdb) set listsize 20
```

现在，您可以通过运行 list 命令且没有额外参数来显示要调试的文件的整个源代码：

```
(gdb) list
1       #include <stdio.h>
2       #include <limits.h>
3
4       int main (int argc, char *argv[]) {
5         unsigned long int a = 0;
6         unsigned long int b = 1;
```

```
7       unsigned long int sum;
8
9       while (b < LONG_MAX) {
10        printf("%ld ", b);
11        sum = a + b;
12        a = b;
13        b = sum;
14      }
15
16      return 0;
17    }
```

8.5. 设置断点

设置新明点

在特定行上设置一个新的断点：

```
(gdb) break file_name:line_number
```

您还可以在某个功能上设置断点：

```
(gdb) break file_name:function_name
```

例 8.5. 设置新明点

这个示例假设您已使用调试信息编译了在 例 8.1 “使用调试信息编译 C 程序” 中列出的 fibonacci.c 文件。

在第 10 行设置一个新的断点：

```
(gdb) break 10
Breakpoint 1 at 0x4004e5: file fibonacci.c, line 10.
```

列出断点

显示当前设置的 breakpoints 列表：

```
(gdb) info breakpoints
```

例 8.6. 列出断点

这个示例假设您已遵循 例 8.5 “设置新明点” 中的说明。

显示当前设置的 breakpoints 列表：

```
(gdb) info breakpoints
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x00000000004004e5 in main at fibonacci.c:10
```

删除现有明点

要删除在某个行中设置的断点：

```
(gdb) clear line_number
```

同样，要删除在特定功能中设置的断点：

```
(gdb) clear function_name
```

例 8.7. 删除现有的明点

这个示例假设您已使用调试信息编译了在 例 8.1 “使用调试信息编译 C 程序” 中列出的 fibonacci.c 文件。

在第 7 行设置一个新的断点：

```
(gdb) break 7
Breakpoint 2 at 0x4004e3: file fibonacci.c, line 7.
```

删除这个断点：

```
(gdb) clear 7
Deleted breakpoint 2
```

8.6. 启动执行

要开始执行要调试的程序：

```
(gdb) run
```

如果程序接受任何命令行参数，您可以将它们作为参数提供给 run 命令：

```
(gdb) run argument...
```

当达到第一个断点（若有）时，执行将停止，当发生错误或程序终止时，执行将停止。

例 8.8. 执行 fibonacci Binary 文件

这个示例假设您已遵循 例 8.5 “设置新明点” 中的说明。

执行 fibonacci 二进制文件：

```
(gdb) run
Starting program: /home/john/fibonacci

Breakpoint 1, main (argc=1, argv=0x7fffffffe4d8) at fibonacci.c:10
10          printf("%ld ", b);
```

8.7. 显示当前值

gdb 实用程序允许您显示与程序相关的几乎任何事物的值，从任何复杂性到有效的表达式，甚至库函数。但是，最常见的任务是显示变量的值。

显示特定变量的当前值：

```
(gdb) print variable_name
```

例 8.9. 显示变量的当前值

本例假设您已遵循 例 8.8 “执行 fibonacci Binary 文件” 中的说明，在到达断点第 10 行后停止了 fibonacci 二进制文件。

显示变量 a 和 b 的当前值：

```
(gdb) print a
$1 = 0
(gdb) print b
$2 = 1
```

8.8. 继续执行

在到达断点后恢复您要进行调试的程序执行：

```
(gdb) continue
```

当到达另一个断点时，执行会再次停止。要跳过一定数量的断点（通常在调试循环时）：

```
(gdb) continue number
```

gdb 工具还允许您在执行单行代码后停止执行：

```
(gdb) step
```

最后，您可以执行一定数量的行：

```
(gdb) step number
```

例 8.10. 继续执行 fibonacci Binary 文件

本例假设您已遵循 例 8.8 “执行 fibonacci Binary 文件” 中的说明，在到达断点第 10 行后停止了 fibonacci 二进制文件。

恢复执行：

```
(gdb) continue
Continuing.

Breakpoint 1, main (argc=1, argv=0x7fffffffe4d8) at fibonacci.c:10
10          printf("%ld ", b);
```

下一次到达断点时，执行将停止。

执行接下来的三行：

```
(gdb) step 3
13          b = sum;
```

这样，您可以在将其分配给 b 之前验证 sum 变量的当前值：

```
(gdb) print sum
$3 = 2
```

8.9. 其它资源

有关 GNU Debugger 及其功能的更多信息，请参阅以下列出的资源。

安装的文档

安装 devtoolset-11-gdb-doc 软件包提供了 HTML 和 PDF 格式的以下文档，采用 /opt/rh/devtoolset-11/root/usr/share/doc/devtoolset-11-gdb-doc-10.2 目录：

- 使用 GDB 书进行调试，这是同名上游资料的副本。本文档的版本与 Red Hat Developer Toolset 提供的 GDB 版本完全对应。
- GDB 的 Obsolete Annotations 文档，它列出了过时的 GDB 级别 2 注解。

在线文档

- Red Hat Enterprise Linux 7 开发人员指南 - Red Hat Enterprise Linux 7 开发人员指南 提供了 GNU Debugger 和调试的更多信息。
- GDB 文档 - 上游 GDB 文档包括 GDB 用户手册和其他参考材料。

另请参阅

- 第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系

统中安装它的更多信息。

- *第 2 章 GNU Compiler Collection(GCC) - 有关如何编译在 C、 C++ 和 Fortran 中编写的程序的更多信息。*

- *第 9 章 strace - 使用 strace 实用程序监控程序使用和接收信号的系统调用的说明。*

- *第 11 章 memstomp - 使用 memstomp 程序识别对带有不同标准不允许重叠的内存区域的库功能的调用的说明。*

第 9 章 STRACE

strace 是命令行的诊断和调试工具，可用于跟踪由正在运行的进程发出和接收的系统调用。它记录每个系统调用的名称、其参数及其返回值，以及进程和其他与内核交互的信号，并将这些记录输出到标准输出或所选文件。

Red Hat Developer Toolset 带有 strace 5.13。

9.1. 安装 STRACE

在 Red Hat Enterprise Linux 中，strace 程序由 devtoolset-11-strace 软件包提供，并使用 devtoolset-11-toolchain 自动安装，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

9.2. 使用 STRACE

要在您要分析的程序中运行 strace 工具：

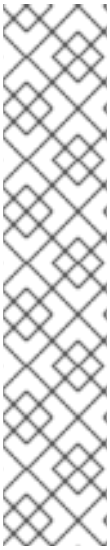
```
$ scl enable devtoolset-11 'strace program argument...'
```

使用您要 分析的程序的名称替换 program，使用您要提供的任何命令行选项和参数替换 program。或者，您可以使用 -p 命令行选项以及进程 ID 在已经运行的进程中运行实用程序：

```
$ scl enable devtoolset-11 'strace -p process_id'
```

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset strace 作为默认值运行 shell 会话：

```
$ scl enable devtoolset-11 'bash'
```



注意

在任意时间点上验证您使用的 strace 版本：

```
$ which strace
```

Red Hat Developer Toolset 的 strace 可执行路径将以 /opt 开头。另外，您可以使用以下命令确认与 Red Hat Developer Toolset strace 的版本号匹配：

```
$ strace -V
```

9.2.1. 将输出重定向到文件

默认情况下，strace 会打印每个系统调用的名称，其参数和返回值到标准错误输出。要将这个输出重定向到文件，请使用 -o 命令行选项以及文件名：

```
$ scl enable devtoolset-11 'strace -o file_name program argument...'
```

使用文件名替换 file_name。

例 9.1. 将输出重定向到文件

考虑从 例 8.1 “使用调试信息编译 C 程序” 的 fibonacci 文件的稍加修改版本。这个可执行文件显示 Fibonacci 序列，并可以选择指定要列出此序列的多少个成员。在此文件上运行 strace 工具，并将 trace 输出重定向到 fibonacci.log ：

```
$ scl enable devtoolset-11 'strace -o fibonacci.log ./fibonacci 20'
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

这会在当前工作目录中创建一个名为 fibonacci.log 的纯文本文件。

9.2.2. 跟踪所选系统调用

要只追踪所选系统调用集合，请使用 -e 命令行选项运行 strace 工具：

```
$ scl enable devtoolset-11 'strace -e expression program argument...'
```

使用以逗号分隔的系统调用列表替换 expression，用于跟踪或 表 9.1 “常用的 -e 选项的值” 中列出的任何关键字。有关所有可用值的详细描述，请查看 strace(1)手册页。

表 9.1. 常用的 -e 选项的值

值	描述
%file	接受文件名作为参数的系统调用。
%process	与进程管理相关的系统调用。
%network	与网络相关的系统调用。
%signal	与信号管理相关的系统调用。
%IPC	与进程间通信相关的系统调用(IPC)。
%desc	与文件描述符相关的系统调用。

请注意，语法 -e 表达式 是完整形式的简写格式 -e trace=表达式。

例 9.2. 跟踪所选系统调用

考虑 例 11.1 “使用 memstomp” 中的 员工 文件。在此可执行文件上运行 strace 工具，仅跟踪 mmap 和 munmap 系统调用：

```
$ scl enable devtoolset-11 'strace -e mmap,munmap ./employee'
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f896c744000
mmap(NULL, 61239, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f896c735000
mmap(0x3146a00000, 3745960, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x3146a00000
mmap(0x3146d89000, 20480, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x189000) = 0x3146d89000
mmap(0x3146d8e000, 18600, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x3146d8e000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f896c734000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f896c733000
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f896c732000
munmap(0x7f896c735000, 61239)           = 0
mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f896c743000
John,john@example.comDoe,
+++ exited with 0 +++
```

-

9.2.3. 显示时间戳

要在 trace 的每行中加上精确的时间（以小时、分钟和秒为单位），请使用 -t 命令行选项运行 strace 工具：

```
$ scl enable devtoolset-11 'strace -t program argument...'
```

要显示毫秒，请提供 -t 选项两次：

```
$ scl enable devtoolset-11 'strace -tt program argument...'
```

要使用执行相应系统调用所需的时间为 trace 的每一行添加前缀，请使用 -r 命令行选项：

```
$ scl enable devtoolset-11 'strace -r program argument...'
```

例 9.3. 显示时间戳

考虑名为 pwd 的可执行文件。在此文件上运行 strace 工具，并在输出中包括时间戳：

```
$ scl enable devtoolset-11 'strace -tt pwd'
19:43:28.011815 execve("./pwd", ["./pwd"], [/* 36 vars */]) = 0
19:43:28.012128 brk(0)                  = 0xcd3000
19:43:28.012174 mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fc869cb0000
19:43:28.012427 open("/etc/ld.so.cache", O_RDONLY) = 3
19:43:28.012446 fstat(3, {st_mode=S_IFREG|0644, st_size=61239, ...}) = 0
19:43:28.012464 mmap(NULL, 61239, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fc869ca1000
19:43:28.012483 close(3)                = 0
...
19:43:28.013410 +++ exited with 0 +++
```

9.2.4. 显示概述

要显示执行每个系统调用所需的时间摘要、执行这些系统调用的次数以及执行期间遇到的错误数量，请使用 -c 命令行选项运行 strace 工具：

```
$ scl enable devtoolset-11 'strace -c program argument...'
```

例 9.4. 显示概述

考虑名为 lsblk 的可执行文件。在此文件中运行 strace 工具并显示 trace 概述：

```
$ scl enable devtoolset-11 'strace -c lsblk > /dev/null'
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 80.88    0.000055           1       106        16 open
 19.12    0.000013           0       140           munmap
  0.00    0.000000           0       148           read
  0.00    0.000000           0         1           write
  0.00    0.000000           0       258           close
  0.00    0.000000           0        37         2 stat
...
------ ----------- ----------- --------- --------- ----------------
100.00    0.000068                  1790        35 total
```

9.2.5. 使用系统调用结果修改

模拟从系统调用返回的错误可帮助识别程序中缺少的错误处理。

要使程序在特定系统调用时收到通用错误，请使用 -e fault= 选项运行 strace 工具并提供系统调用：

```
$ scl enable devtoolset-11 'strace -e fault=syscall program argument...'
```

要指定错误类型或返回值，请使用 -e inject= 选项：

```
$ scl enable devtoolset-11 'strace -e inject=syscall:error=error-type program argument'
$ scl enable devtoolset-11 'strace -e inject=syscall:retval=return-value program argument'
```

请注意，指定错误类型和返回值是互斥的。

例 9.5. 使用系统调用结果修改

考虑名为 lsblk 的可执行文件。在这个文件中运行 strace 工具，使 mmap（） 系统调用返回错误：

```
$ scl enable devtoolset-11 'strace -e fault=mmap:error=EPERM lsblk > /dev/null'
execve("/usr/bin/lsblk", ["lsblk"], 0x7fff1c0e02a0 /* 54 vars */) = 0
brk(NULL)                               = 0x55d9e8b43000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
-1 EPERM (Operation not permitted) (INJECTED)
writev(2, [{iov_base="lsblk", iov_len=5}, {iov_base=": ", iov_len=2}, {iov_base="error while
```

```
loading shared libra"..., iov_len=36}, {iov_base=": ", iov_len=2}, {iov_base="", iov_len=0},
{iov_base="", iov_len=0}, {iov_base="cannot create cache for search p"..., iov_len=35},
{iov_base=": ", iov_len=2}, {iov_base="Cannot allocate memory", iov_len=22},
{iov_base="\n", iov_len=1}], 10lsblk: error while loading shared libraries: cannot create
cache for search path: Cannot allocate memory
) = 105
exit_group(127)                         = ?
+++ exited with 127 +++
```

9.3. 其它资源

有关 strace 及其功能的更多信息，请参阅以下列出的资源。

安装的文档

- strace(1)- strace 实用程序手册页面提供有关其使用情况的详细信息。显示 Red Hat Developer Toolset 中包含的版本的 man page：

```
$ scl enable devtoolset-11 'man strace'
```

另请参阅

- 第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。
- 第 10 章 ltrace - 使用 ltrace 工具追踪程序库调用的说明。
- 第 8 章 GNU 调试器(GDB) - 关于在 C、C++ 和 Fortran 中编写的调试程序的说明。
- 第 11 章 memstomp - 使用 memstomp 程序识别对带有不同标准不允许重叠的内存区域的库功能的调用的说明。

第 10 章 LTRACE

ltrace 是命令行的诊断和调试工具，可用于显示对共享库进行的调用。它使用动态库 hooking 机制，可防止它跟踪调用静态链接的库。ltrace 还显示库调用的返回值。输出将打印到标准输出或所选文件。

Red Hat Developer Toolset 带有 ltrace 0.7.91。虽然基础版本 ltrace 与之前 Red Hat Developer Toolset 版本相同，但各种改进和程序错误修复已被移植。

10.1. 安装 LTRACE

在 Red Hat Enterprise Linux 中，ltrace 程序由 devtoolset-11-ltrace 软件包提供，并使用 devtoolset-11-toolchain 自动安装，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

10.2. 使用 LTRACE

要在您要分析的程序中运行 ltrace 工具：

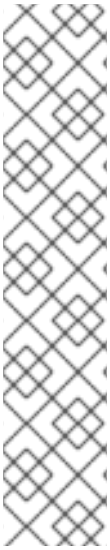
```
$ scl enable devtoolset-11 'ltrace program argument...'
```

使用您要 分析的程序的名称替换 program，使用您要提供的任何命令行选项和参数替换 program。或者，您可以使用 -p 命令行选项以及进程 ID 在已经运行的进程中运行实用程序：

```
$ scl enable devtoolset-11 'ltrace -p process_id'
```

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您以 Red Hat Developer Toolset ltrace 作为默认值运行 shell 会话：

```
$ scl enable devtoolset-11 'bash'
```



注意

在任意时间点验证您使用的 ltrace 版本：

```
$ which ltrace
```

Red Hat Developer Toolset 的 ltrace 可执行文件路径以 /opt 开头。另外，您可以使用以下命令确认与 Red Hat Developer Toolset ltrace 的版本号匹配：

```
$ ltrace -V
```

10.2.1. 将输出重定向到文件

默认情况下，l trace 会打印每个系统调用的名称，其参数和返回值到标准错误输出。要将这个输出重定向到文件，请使用 -o 命令行选项以及文件名：

```
$ scl enable devtoolset-11 'ltrace -o file_name program argument...'
```

使用文件名替换 file_name。

例 10.1. 将输出重定向到文件

考虑从 *例 8.1 “使用调试信息编译 C 程序”* 的 fibonacci 文件的稍加修改版本。这个可执行文件显示 Fibonacci 序列，并可以选择指定要列出此序列的多少个成员。在此文件上运行 ltrace 实用程序并将 trace 输出重定向到 fibonacci.log ：

```
$ scl enable devtoolset-11 'ltrace -o fibonacci.log ./fibonacci 20'
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
```

这会在当前工作目录中创建一个名为 fibonacci.log 的纯文本文件。

10.2.2. 跟踪所选库调用

要只跟踪一组所选库调用，请使用 -e 命令行选项运行 ltrace 工具：

```
$ scl enable devtoolset-11 'ltrace -e expression program argument...'
```

使用规则链替换 expression，以指定要跟踪的库调用。规则可由标识符号名称的模式组成（如 malloc 或 free）和标识库 SONAME（如 libc.so）的模式。例如，要跟踪调用 malloc 和 free 功能，但要省略 libc 库所完成的那些功能：

```
$ scl enable devtoolset-11 'ltrace -e malloc+free-@libc.so* program'
```

例 10.2. 跟踪所选库调用

考虑 ls 命令。在此程序上运行 ltrace 程序，只跟踪 opendir、readdir、和 closedir 功能调用：

```
$ scl enable devtoolset-11 'ltrace -e opendir+readdir+closedir ls'
ls->opendir(".")    = { 3 }
ls->readdir({ 3 })  = { 61533, "." }
ls->readdir({ 3 })  = { 131, ".." }
ls->readdir({ 3 })  = { 67185100, "BUILDROOT" }
ls->readdir({ 3 })  = { 202390772, "SOURCES" }
ls->readdir({ 3 })  = { 60249, "SPECS" }
ls->readdir({ 3 })  = { 67130110, "BUILD" }
ls->readdir({ 3 })  = { 136599168, "RPMS" }
ls->readdir({ 3 })  = { 202383274, "SRPMS" }
ls->readdir({ 3 })  = nil
ls->closedir({ 3 }) = 0
BUILD  BUILDROOT  RPMS  SOURCES  SPECS  SRPMS
+++ exited (status 0) +++
```

有关可用过滤表达式的详细描述，请参阅 ltrace(1)手册页。

10.2.3. 显示时间戳

要在 trace 的每行中加上精确的时间（以小时、分钟和秒为单位），请使用 -t 命令行选项运行 ltrace 实用程序：

```
$ scl enable devtoolset-11 'ltrace -t program argument...'
```

要显示毫秒，请提供 -t 选项两次：

```
$ scl enable devtoolset-11 'ltrace -tt program argument...'
```

要使用执行相应系统调用所需的时间为 trace 的每一行添加前缀，请使用 -r 命令行选项：

```
$ scl enable devtoolset-11 'ltrace -r program argument...'
```

例 10.3. 显示时间戳

考虑 **pwd** 命令。在此程序上运行 **ltrace** 工具，并在输出中包含时间戳：

```
$ scl enable devtoolset-11 'ltrace -tt pwd'
13:27:19.631371 __libc_start_main([ "pwd" ] <unfinished ...>
13:27:19.632240 getenv("POSIXLY_CORRECT")                    = nil
13:27:19.632520 strrchr("pwd", '/')                          = nil
13:27:19.632786 setlocale(LC_ALL, "")                        = "en_US.UTF-8"
13:27:19.633220 bindtextdomain("coreutils", "/usr/share/locale") = "/usr/share/locale"
13:27:19.633471 textdomain("coreutils")                      = "coreutils"
(...)
13:27:19.637110  exited (status 0)
```

10.2.4. 显示概述

要显示执行每个系统调用以及执行这些系统调用的次数，请使用 **-c** 命令行选项运行 **ltrace** 工具：

```
$ scl enable devtoolset-11 'ltrace -c program argument...'
```

例 10.4. 显示概述

考虑 **lsblk** 命令。在此程序上运行 **ltrace** 工具并显示 **trace** 概述：

```
$ scl enable devtoolset-11 'ltrace -c lsblk > /dev/null'
% time     seconds  usecs/call     calls      function
------ ----------- ----------- --------- --------------------
 53.60    0.261644      261644         1 __libc_start_main
  4.48    0.021848          58       374 mbrtowc
  4.41    0.021524          57       374 wcwidth
  4.39    0.021409          57       374 __ctype_get_mb_cur_max
  4.38    0.021359          57       374 iswprint
  4.06    0.019838          74       266 readdir64
  3.21    0.015652          69       224 strlen
...
------ ----------- ----------- --------- --------------------
100.00    0.488135                  3482 total
```

10.3. 其它资源

有关 ltrace 及其功能的更多信息，请参阅以下列出的资源。

安装的文档

- ltrace(1)- ltrace 实用程序的 man page 提供了有关其使用情况的详细信息。显示 Red Hat Developer Toolset 中包含的版本的 man page：

```
$ scl enable devtoolset-11 'man ltrace'
```

在线文档

- RHEL 6 和 7 的 ltrace - Red Hat Developer Blog 中的这篇文章提供了有关如何使用 ltrace 进行应用程序调试的其他深度信息（包括实际示例）。

另请参阅

- 第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。
- 第 9 章 strace - 使用 strace 工具跟踪程序系统调用的说明。
- 第 8 章 GNU 调试器(GDB) - 关于在 C、C++ 和 Fortran 中编写的调试程序的说明。
- 第 11 章 memstomp - 使用 memstomp 程序识别对带有不同标准不允许重叠的内存区域的库功能的调用的说明。

第 11 章 MEMSTOMP

memstomp 是一个命令行工具，可用于识别当各种标准不允许重叠时具有重叠内存区域的函数调用。它截获对 表 11.1 “由memstomp 表示的函数调用” 中列出的库功能以及每个内存重叠的调用，它会显示详细的回溯追踪以帮助您调试问题。

与 Valgrind 类似，memstomp 程序会检查应用程序，而无需重新编译它们。但是，这个工具比这个工具快得多，因此作为一种方便的替代方案。

Red Hat Developer Toolset 带有 memstomp 0.1.5。

表 11.1. 由memstomp 表示的函数调用

功能	描述
memcpy	将一个内存区域中的 *n* 字节复制到另一个内存区域，并返回指向第二个内存区域的指针。
memccpy	将最多 *n* 字节从一个内存区域复制到另一个内存区域，并在找到特定字符时停止。它将指针返回到最后写入字节的字节，如果找不到给定字符，则返回 NULL。
mempcpy	将一个内存区的 *n* 字节复制到另一个内存区域，并在最后写入字节后返回指针到字节。
strcpy	将字符串从一个内存区域复制到另一个内存区域，并返回指向第二个字符串的指针。
stpcpy	将字符串从一个内存区域复制到另一个内存区域，并将指针返回到第二个字符串的终止空字节。
strncpy	将一个字符串最多 *n* 个字符复制到另一个字符串，并将指针返回到第二个字符串。
stpncpy	将一个字符串最多 *n* 个字符复制到另一个字符串。它将指针返回到第二个字符串的终止空字节，或者如果字符串没有 nullterminated，则会在最后写的字节后面指向指向字节的指针。
strcat	在覆盖第二个字符串的终止 null 字节并在其末尾添加新字符串时，将一个字符串附加到另一个字符串。它将一个指向新字符串的指针。
strncat	在覆盖第二个字符串的终止 null 字节并在其末尾添加新字符串时，把最多 *n* 个字符附加到另一个字符串。它将一个指向新字符串的指针。
wmemcpy	与 **memcpy**（）函数相当的字符，可将 *n* wide 字符从一个数组复制到另一个阵列，并将指针返回到第二个数组。
wmempcpy	等效于 **mempcpy**（）函数的字符，它可将 *n* wide 字符从一个阵列复制到另一个阵列，并在最后写的字符后将指针返回到字节。

功能	描述
wcscpy	类似于 **strcpy**（）函数的字符等式字符可将一个数组中的字符字符串复制到另一个阵列，并将指针返回到第二个数组。
wcsncpy	类似于 **strncpy**（）函数的字符最大的字符从一个阵列复制到另一个阵列，并将指针返回到第二个字符串。
wcscat	类似于 **strcat**（）函数的字符等字符可向另一个字符字符串附加一个广泛的字符字符串，同时覆盖第二个字符串的终止 null 字节，并在其末尾添加一个新字符串。它将一个指向新字符串的指针。
wcsncat	等效于 **strncat**（）函数的字符可连接一个阵列最多的 *n* 个字符，同时覆盖第二个广字符字符串的终止的空字节并在其末尾添加一个新的字符。它将一个指向新字符串的指针。

11.1. 安装 MEMSTOMP

在 Red Hat Developer Toolset 中，memstomp 工具由 devtoolset-11-memstomp 软件包提供，并会自动安装 devtoolset-11-toolchain，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

11.2. 使用 MEMSTOMP

要在您要分析的程序中运行 memstomp 程序：

```
$ scl enable devtoolset-11 'memstomp program argument...'
```

要在检测到问题时立即终止分析的程序，请使用 --kill （或 -k 表示短）命令行选项运行实用程序：

```
$ scl enable devtoolset-11 'memstomp --kill program argument...'
```

最好使用 --kill 选项。如果您正在分析多线程程序；后端的内部实施不是 thread-safe，并在多线程程序上运行 memstomp 程序，因此无需此命令行选项即可生成不可靠的结果。

另外，如果使用调试信息或者这个调试信息编译了分析的程序，您可以使用 --debug-info （或 -d）命令行选项来生成更详细的后端：

```
$ scl enable devtoolset-11 'memstomp --debug-info program argument...'
```

有关如何使用二进制文件中内置的调试信息编译程序的详情，请参考 第 8.2 节 “准备进行调试的程序”。有关如何为任何 Red Hat Developer Toolset 软件包安装调试信息的详情，请参考 第 1.5.4 节 “安装调试信息”。

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset memstomp 运行 shell 会话作为默认值：

```
$ scl enable devtoolset-11 'bash'
```

例 11.1. 使用 memstomp

在当前工作目录中，创建名为 staff .c 的 源文件，其内容如下：

```
#include <stdio.h>
#include <string.h>

#define BUFSIZE 80

int main(int argc, char *argv[]) {
  char employee[BUFSIZE] = "John,Doe,john@example.com";
  char name[BUFSIZE] = {0};
  char surname[BUFSIZE] = {0};
  char *email;
  size_t length;

  /* Extract the information: */
  memccpy(name, employee, ',', BUFSIZE);
  length = strlen(name);
  memccpy(surname, employee + length, ',', BUFSIZE);
  length += strlen(surname);
  email = employee + length;

  /* Compose the new entry: */
  strcat(employee, surname);
  strcpy(employee, name);
  strcat(employee, email);

  /* Print the result: */
  puts(employee);

  return 0;
}
```

将这个程序编译到名为 employees 的二进制文件 中 ：

```
$ scl enable devtoolset-11 'gcc -rdynamic -g -o employee employee.c'
```

识别带有重叠内存区域的错误功能调用：

```
$ scl enable devtoolset-11 'memstomp --debug-info ./employee'
memstomp: 0.1.4 successfully initialized for process employee (pid 14887).

strcat(dest=0x7fff13afc265, src=0x7fff13afc269, bytes=21) overlap for employee(14887)
        ??:0    strcpy()
        ??:0    strcpy()
        ??:0    _Exit()
        ??:0    strcat()
        employee.c:26   main()
        ??:0    __libc_start_main()
        ??:0    _start()
John,john@example.comDoe,
```

11.3. 其它资源

有关 memstomp 及其功能的更多信息，请参阅以下列出的资源。

安装的文档

- memstomp(1)- memstomp 实用程序的 man page 提供了有关其使用情况的详细信息。显示 Red Hat Developer Toolset 中包含的版本的 man page：

  ```
  $ scl enable devtoolset-11 'man memstomp'
  ```

另请参阅

- 第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。
- 第 8 章 GNU 调试器(GDB) - 关于在 C、C++ 和 Fortran 中编写的调试程序的说明。
- 第 9 章 strace - 使用 strace 实用程序监控程序使用和接收信号的系统调用的说明。
- 第 13 章 Valgrind - 使用 Valgrind 工具分析应用程序并检测内存错误和内存管理问题的说明，如使用未初始化内存、分配和释放内存，以及在系统调用中使用不正确的参数。

部分 IV. 性能监控工具

第 12 章 SYSTEMTAP

SystemTap 是跟踪和探测工具，允许用户监控整个系统的活动，而无需检测、重新编译、安装和重新引导。它可以通过自定义脚本语言进行编程，该语言可为其提供表达性（追踪、过滤和分析），并到达（查找正在运行的内核和应用程序）。

SystemTap 可以监控各种类型的事件，如内核或应用程序中的函数调用、计时器、追踪点、性能计数器等。某些包括的示例脚本会生成与 netstat、ps、top 和 iostat 类似的输出，其他脚本包括用于解决安全问题的广泛打印函数调用图形或工具。

Red Hat Developer Toolset 带有 SystemTap 4.5。此版本比 Red Hat Developer Toolset 中包含的版本更新，提供多个程序错误修复和增强。

表 12.1. 红帽开发人员工具集通过 SystemTap 分发的工具

Name	描述
stap	将指令转换为 C 代码，构建内核模块并将其加载到正在运行的 Linux 内核中。
stapdyn	SystemTap 的 Dyninst 后端。
staprun	从使用 **stap** 实用程序构建的内核模块加载、卸载、附加和分离。
stapsh	作为 SystemTap 的远程 shell。
stap-prep	确定 和-if possible-downloads 运行 SystemTap 所需的内核信息软件包。
stap-merge	合并每个 CPU 文件。当使用 **-b** 命令行选项执行 **stap** 实用工具时，该脚本会自动执行。
stap-report	收集系统的重要信息，以报告 SystemTap 中的错误。
stap-server	编译服务器，用于侦听来自 **stap** 客户端的请求。

12.1. 安装 SYSTEMTAP

在红帽 Developer Toolset 中，devtoolset-11-systemtap 软件包提供 SystemTap，并自动安装 devtoolset-11-perftools，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

为了将工具放入 Linux 内核，SystemTap 可能还需要使用调试信息安装附加软件包。要确定要安装的软件包，请运行 stap-prep 工具，如下所示：

```
$ scl enable devtoolset-11 'stap-prep'
```

请注意，如果您以 root 用户身份执行此命令，实用程序会自动提供用于安装的软件包。有关如何在系统中安装这些软件包的更多信息，请参阅 Red Hat Enterprise Linux 7 SystemTap 开始使用指南。

12.2. 使用 SYSTEMTAP

执行属于 SystemTap 的任何工具：

```
$ scl enable devtoolset-11 'tool option...'
```

有关使用 SystemTap 分发的工具列表，请参阅 表 12.1 “红帽开发人员工具集通过 SystemTap 分发的工具”。例如，运行 stap 工具来构建检测模块：

```
$ scl enable devtoolset-11 'stap option... argument...'
```

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset SystemTap 作为默认值运行 shell 会话：

```
$ scl enable devtoolset-11 'bash'
```

注意

在任何时候验证您使用的 SystemTap 版本：

```
$ which stap
```

Red Hat Developer Toolset 的 stap 可执行路径将以 /opt 开头。另外，您可以使用以下命令确认与 Red Hat Developer Toolset SystemTap 的版本号匹配：

```
$ stap -V
```

12.3. 其它资源

有关 SystemTap 及其功能的更多信息，请参见下面列出的资源。

安装的文档

- stap(1)- stap 命令的 man page 提供了对其使用情况的详细信息，并引用其他相关的 man page。显示 Red Hat Developer Toolset 中包含的版本的 man page：

```
$ scl enable devtoolset-11 'man stap'
```

- staprun(8)- staprun 命令的 man page 提供了有关其使用情况的详细信息。显示 Red Hat Developer Toolset 中包含的版本的 man page：

```
$ scl enable devtoolset-11 'man staprun'
```

在线文档

- Red Hat Enterprise Linux 7 SystemTap 发起者指南 - 用于 Red Hat Enterprise Linux 7 的 SystemTap 入门指南，提供 SystemTap 及其用法简介。

- Red Hat Enterprise Linux 7 SystemTap Tapset Reference - Red Hat Enterprise Linux 7 的 SystemTap Tapset 参考 提供了有关 SystemTap 的进一步详情。

- SystemTap 文档 - SystemTap 文档提供了更多有关 SystemTap 文档，以及 SystemTap 脚本的许多示例。

另请参阅

- 第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。

- 第 13 章 Valgrind - 使用 Valgrind 工具分析应用程序并检测内存错误和内存管理问题的说明，如使用未初始化内存、分配和释放内存，以及在系统调用中使用不正确的参数。

- 第 14 章 OProfile - 使用 OProfile 工具确定哪个部分代码消耗最大 CPU 时间以及原因。

- 第 15 章 Dyninst - 使用 Dyninst 库检测用户空间可执行文件的说明。

第 13 章 VALGRIND

Valgrind 是一个工具框架，随附一些工具来分析应用程序。它可用于检测各种内存错误和内存管理问题，如使用未初始化内存或内存不足的分配和释放内存，或者识别系统调用中不正确的参数。如需通过 Red Hat Developer Toolset 版本的 Valgrind 分发的性能分析工具的完整列表，请参阅 表 13.1 “使用 Red Hat Developer Toolset 的 Valgrind 分发工具”。

Valgrind 配置集会重新编写应用程序并检测重写的二进制代码。这可让您在不重新编译应用程序的情况下对应用程序进行性能分析，但它也比其他配置集器要慢得多，特别是在执行非常详细的运行时时。 因此，它不适用于调试特定时间的问题或内核空间调试。

Red Hat Developer Toolset 带有 Valgrind 3.17.0。此版本比 Red Hat Developer Toolset 中包含的版本更新，提供多个程序错误修复和增强。

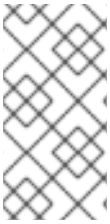
表 13.1. 使用 Red Hat Developer Toolset 的 Valgrind 分发工具

Name	描述
Memcheck	通过截获系统调用并检查所有读写操作，检测内存管理问题。
cachegrind	通过模拟 1 指令缓存(I1)、级别 1 数据缓存(D1)和统一级 2 缓存(L2)来识别缓存丢失的来源。
Callgrind	生成代表函数调用历史记录的调用图形。
Helgrind	检测多线程 C、C++ 和 Fortran 程序使用 POSIX 线程原语的同步错误。
DRD	检测多线程 C 和 C++ 程序中的错误，它们使用 POSIX 线程原语或其他在 POSIX 线程原语基础上构建的任何其他线程概念。
Massif	监控堆和堆栈使用情况。

13.1. 安装 VALGRIND

在 Red Hat Developer Toolset 中， Vrind 由 devtoolset-11-valgrind 软件包提供，并会自动安装 devtoolset-11-perftools。

有关如何在系统中安装 Red Hat Developer Toolset 和相关软件包的详情，请参考 第 1.5 节 “安装 Red Hat Developer Toolset”。



注意

请注意，如果您结合使用 Valgrind 和 GNU Debugger，建议您使用 Red Hat Developer Toolset 中包含的 GDB 版本，以确保完全支持所有功能。

13.2. 使用 VALGRIND

要在您要配置集的程序中运行任何 Valgrind 工具：

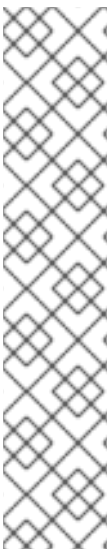
```
$ scl enable devtoolset-11 'valgrind --tool=tool program argument...'
```

如需使用 Valgrind 分发的工具列表，请参阅 表 13.1 “使用 Red Hat Developer Toolset 的 Valgrind 分发工具”。--tool 命令行选项的参数必须以小写方式指定，如果省略这个选项，则默认使用 Memcheck。例如，要在程序中运行 Cachegrind 来识别缓存丢失的来源：

```
$ scl enable devtoolset-11 'valgrind --tool=cachegrind program argument...'
```

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset Valgrind 运行 shell 会话，以默认方式运行：

```
$ scl enable devtoolset-11 'bash'
```



注意

验证您要使用的 Valgrind 版本：

```
$ which valgrind
```

Red Hat Developer Toolset 的 valgrind 可执行路径从 /opt 开始。另外，您可以使用以下命令确认与 Red Hat Developer Toolset Valgrind 匹配的版本号：

```
$ valgrind --version
```

13.3. 其它资源

有关 Valgrind 及其功能的更多信息，请参阅以下列出的资源。

安装的文档

- *Valgrind(1)- valgrind 实用程序手册页面提供了有关如何使用 Valgrind 的详细信息。显示 Red Hat Developer Toolset 中包含的版本的 man page：*

```
$ scl enable devtoolset-11 'man valgrind'
```

- *Valgrind 文档 - Valgrind 的 HTML 文档位于 /opt/rh/devtoolset-11/root/usr/share/doc/devtoolset-11-valgrind-3.17.0/html/index.html。*

在线文档

- *Red Hat Enterprise Linux 7 开发人员指南 - Red Hat Enterprise Linux 7 开发人员指南 提供了有关 Valgrind 及其 Eclipse 插件的更多信息。*

- *Red Hat Enterprise Linux 7 性能调优指南 - Red Hat Enterprise Linux 7 的性能调优 指南 提供了使用 Valgrind 分析应用程序的更多详情。*

另请参阅

- *第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。*

- *第 11 章 memstomp - 使用 memstomp 程序识别对带有不同标准不允许重叠的内存区域的库功能的调用的说明。*

- *第 12 章 SystemTap - SystemTap 工具介绍，以及如何使用它监控正在运行的系统的活动。*

- *第 14 章 OProfile - 使用 OProfile 工具确定哪个部分代码消耗最大 CPU 时间以及原因。*

- *第 15 章 Dyninst - 使用 Dyninst 库检测用户空间可执行文件的说明。*

第 14 章 OPROFILE

OProfile 是一个低开销，系统范围配置集，使用处理器上的 performance-monitoring 硬件来检索系统中内核和可执行文件的信息，如引用内存、级别 2 缓存(L2)请求的数量以及接收的硬件中断的数量。它包含一个配置实用程序、用于收集数据的守护进程，以及用于将数据转换为人类可读形式的多个工具。有关 Red Hat Developer Toolset 版本所发布的工具的完整列表，请参阅 表 14.1 “使用 OProfile 为 Red Hat Developer Toolset 分发的工具”。

OProfile 配置集了一个应用程序而无需添加任何检测的应用程序，方法是记录每九个事件的详情。这样，它可以消耗比 Valgrind 少的资源，但它也会导致其样本更精确。与 Valgrind 仅收集单个进程及其在用户空间中的子代数据不同，OProfile 非常适合收集用户空间和内核空间进程上的系统范围数据，并且需要 root 特权才能运行。

Red Hat Developer Toolset 带有 OProfile 1.4.0。

表 14.1. 使用 OProfile 为 Red Hat Developer Toolset 分发的工具

Name	描述
operf	使用 Linux 性能事件子系统记录单个进程或系统范围的示例。
opannotate	从性能分析数据生成注解的源文件或装配列表。
oparchive	生成包含可执行、调试和示例文件的目录。
opgprof	生成与 **gprof** 兼容的配置集会话概述。
ophelp	显示可用事件的列表。
opimport	将外部二进制格式中的示例数据库文件转换为原生格式。
opjitconv	将即时(JIT)转储文件转换为可执行文件和可链接格式(ELF)。
opreport	生成性能分析会话的镜像和符号摘要。
ocount	用于计算受监控命令的期间内特定事件发生次数的新工具。

14.1. 安装 OPROFILE

在 Red Hat Developer Toolset 中，OProfile 由 devtoolset-11-oprofile 软件包提供，并自动安装 devtoolset-11-perftools，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。

14.2. 使用 OPROFILE

运行任何由 OProfile 分发的工具：

```
# scl enable devtoolset-11 'tool option...'
```

如需 OProfile 分发的工具列表，请参阅 表 14.1 “使用 OProfile 为 Red Hat Developer Toolset 分发的工具”。例如，使用 ophelp 命令以 XML 格式列出可用事件：

```
$ scl enable devtoolset-11 'ophelp -X'
```

请注意，您可以使用 scl 程序执行任何命令，从而导致使用 Red Hat Developer Toolset 二进制文件运行它，而不是 Red Hat Enterprise Linux 系统等同的 Red Hat Enterprise Linux 系统。这可让您使用 Red Hat Developer Toolset OProfile 运行 shell 会话作为默认值：

```
$ scl enable devtoolset-11 'bash'
```

注意

在任意时间点上验证您使用的 OProfile 版本：

```
$ which operf
```

Red Hat Developer Toolset 的 operf 可执行路径从 /opt 开始。另外，您可以使用以下命令确认与 Red Hat Developer Toolset OProfile 的版本号匹配：

```
# operf --version
```

14.3. 其它资源

有关 OProfile 及其功能的更多信息，请参阅以下列出的资源。

安装的文档

- OProfile(1)- 名为 oprofile 的 man page 提供 OProfile 及可用工具的概述。显示 Red Hat Developer Toolset 中包含的版本的 man page：

  ```
  $ scl enable devtoolset-11 'man oprofile'
  ```

- *opannotate(1), oparchive(1), operf(1), opgprof(1), ophelp(1), opimport(1), opreport(1)- Manual page for distributed pages for distributed tools on their respective usage. 显示 Red Hat Developer Toolset 中包含的版本的 man page：*

```
scl enable devtoolset-11 'man tool'
```

在线文档

- *Red Hat Enterprise Linux 7 开发人员指南 - Red Hat Enterprise Linux 7 开发人员指南 提供了有关 OProfile 的更多信息。*

- *Red Hat Enterprise Linux 7 系统管理员指南- Red Hat Enterprise Linux 7 系统管理员指南介绍了如何使用 operf 工具。*

另请参阅

- *第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。*

- *第 12 章 SystemTap - SystemTap 简介以及有关如何使用它监控正在运行的系统活动的说明。*

- *第 13 章 Valgrind - 使用 Valgrind 工具分析应用程序并检测内存错误和内存管理问题的说明，如使用未初始化内存、分配和释放内存，以及在系统调用中使用不正确的参数。*

- *第 15 章 Dyninst - 使用 Dyninst 库检测用户空间可执行文件的说明。*

第 15 章 DYNINST

Dyninst 库提供了一个 应用编程接口 (API)，用于在执行期间使用用户空间可执行文件。它可用于将代码插入到运行程序中，更改特定的子例程调用，甚至将其从程序中删除。它充当重要的调试和性能监控工具。Dyninst API 通常与 SystemTap 一同使用，以允许非root 用户检测用户空间可执行文件。

Red Hat Developer Toolset 带有 Dyninst 11.0.0。

15.1. 安装 DYNINST

在 Red Hat Developer Toolset 中，devtoolset-11-dyninst 软件包提供 Dyninst 库，并使用 devtoolset-11-perftools 自动安装，如 第 1.5 节 “安装 Red Hat Developer Toolset” 所述。另外，建议您安装由 devtoolset-11-toolchain 软件包提供的 GNU Compiler Collection。

如果要为二进制文件编写自定义检测，请安装相关的标头文件：

```
# yum install devtoolset-11-dyninst-devel
```

您还可以安装这个库的 API 文档：

```
# yum install devtoolset-11-dyninst-doc
```

有关 devtoolset-11-dyninst-doc 软件包中所含的文档的完整列表，请参阅 第 15.3 节 “其它资源”。有关如何在系统中安装可选软件包的详情，请参考 第 1.5 节 “安装 Red Hat Developer Toolset”。

15.2. 使用 DYNINST

15.2.1. 使用带有 SystemTap 的 Dyninst

要将 Dyninst 与 SystemTap 结合使用，允许非root 用户检测用户空间可执行文件，使用 --dyninst（或 --runtime=dyninst）命令行选项运行 stap 命令。这会告知 stap 将 SystemTap 脚本转换为使用 Dyninst 库的 C 代码，将这个 C 代码编译到共享库，然后加载共享库并运行脚本。请注意，当执行类似此操作时，stap 命令还需要指定 -c 或 -x 命令行选项。

使用 Dyninst 运行时检测可执行文件：

```
$ scl enable devtoolset-11 "stap --dyninst -c 'command' option... argument..."
```

同样，使用 Dyninst 运行时检测用户的进程：

```
$ scl enable devtoolset-11 "stap --dyninst -x process_id option... argument..."
```

有关 Red Hat Developer Toolset 版本的信息，请参阅 第 12 章 SystemTap 。有关 SystemTap 及其用法的常规介绍，请参见 Red Hat Enterprise Linux 7 的 SystemTap 入门指南。

例 15.1. 使用带有 SystemTap 的 Dyninst

考虑一个名为 exercise.C 的源文件，其内容如下：

```
#include <stdio.h>

void print_iteration(int value) {
  printf("Iteration number %d\n", value);
}

int main(int argc, char **argv) {
  int i;
  printf("Enter the starting number: ");
  scanf("%d", &i);
  for(; i>0; --i)
    print_iteration(i);
  return 0;
}
```

这个程序提示用户输入起始数字，然后计入 1，针对每个迭代调用 print_iteration（） 函数，以便将数字打印到标准输出。使用 Red Hat Developer Toolset 中的 g++ 编译器在命令行中编译该程序：

```
$ scl enable devtoolset-11 'g++ -g -o exercise exercise.C'
```

现在，考虑另一个名为 count.stp 的源文件，其内容如下：

```
#!/usr/bin/stap

global count = 0

probe process.function("print_iteration") {
  count++
}
```

```
probe end {
  printf("Function executed %d times.\n", count)
}
```

此 SystemTap 脚本在执行进程的过程中打印调用 print_iteration（） 函数的次数。在 练习 二进制文件中运行此脚本：

```
$ scl enable devtoolset-11 "stap --dyninst -c './exercise' count.stp"
Enter the starting number: 5
Iteration number 5
Iteration number 4
Iteration number 3
Iteration number 2
Iteration number 1
Function executed 5 times.
```

15.2.2. 使用 Dyninst 作为独立库

在将 Dyninst 库用作应用程序的一部分之前，将 DYNINSTAPI_RT_LIB 环境变量的值设置为到运行时库文件的路径：

```
$ export DYNINSTAPI_RT_LIB=/opt/rh/devtoolset-
11/root/usr/lib64/dyninst/libdyninstAPI_RT.so
```

这会在当前 shell 会话中设置 DYNINSTAPI_RT_LIB 环境变量。

例 15.2 “使用 Dyninst 作为独立应用程序” 演示了如何编写和构建用于监控用户空间进程的执行程序。有关如何使用 Dyninst 的详细解释，请查看 第 15.3 节 “其它资源” 中列出的资源。

例 15.2. 使用 Dyninst 作为独立应用程序

考虑来自 例 15.1 “使用带有 SystemTap 的 Dyninst” 的 exercise.C 源文件：此程序提示用户输入启动号，然后计入 1，为每个迭代调用 print_iteration（） 函数，以便将数字输出到标准输出。

现在，考虑另一个名为 count.C 的源文件，其内容如下：

```
#include <stdio.h>
#include <fcntl.h>
#include "BPatch.h"
#include "BPatch_process.h"
#include "BPatch_function.h"
```

```
#include "BPatch_Vector.h"
#include "BPatch_thread.h"
#include "BPatch_point.h"

void usage() {
  fprintf(stderr, "Usage: count <process_id> <function>\n");
}

// Global information for counter
BPatch_variableExpr *counter = NULL;

void createCounter(BPatch_process *app, BPatch_image *appImage) {
  int zero = 0;
  counter = app->malloc(*appImage->findType("int"));
  counter->writeValue(&zero);
}

bool interceptfunc(BPatch_process *app,
                   BPatch_image *appImage,
                   char *funcName) {
  BPatch_Vector<BPatch_function *> func;
  appImage->findFunction(funcName, func);
  if(func.size() == 0) {
    fprintf(stderr, "Unable to find function to instrument()\n");
    exit (-1);
  }
  BPatch_Vector<BPatch_snippet *> incCount;
  BPatch_Vector<BPatch_point *> *points;
  points = func[0]->findPoint(BPatch_entry);
  if ((*points).size() == 0) {
    exit (-1);
  }

  BPatch_arithExpr counterPlusOne(BPatch_plus, *counter, BPatch_constExpr(1));
  BPatch_arithExpr addCounter(BPatch_assign, *counter, counterPlusOne);

  return app->insertSnippet(addCounter, *points);
}

void printCount(BPatch_thread *thread, BPatch_exitType) {
  int val = 0;
  counter->readValue(&val, sizeof(int));
  fprintf(stderr, "Function executed %d times.\n", val);
}

int main(int argc, char *argv[]) {
   int pid;
   BPatch bpatch;
   if (argc != 3) {
     usage();
     exit(1);
   }
   pid = atoi(argv[1]);
   BPatch_process *app = bpatch.processAttach(NULL, pid);
   if (!app) exit (-1);
   BPatch_image *appImage = app->getImage();
```

```
    createCounter(app, appImage);
    fprintf(stderr, "Finding function %s(): ", argv[2]);
    BPatch_Vector<BPatch_function*> countFuncs;
    fprintf(stderr, "OK\nInstrumenting function %s(): ", argv[2]);
    interceptfunc(app, appImage, argv[2]);
    bpatch.registerExitCallback(printCount);
    fprintf(stderr, "OK\nWaiting for process %d to exit...\n", pid);
    app->continueExecution();
    while (!app->isTerminated())
      bpatch.waitForStatusChange();
    return 0;
  }
```

请注意，在调用任何 Dyninst 库销毁器之前，客户端应用程序应该销毁所有 Bpatch 对象。否则，mutator 可能会意外终止分段错误。要临时解决这个问题，请将 mutator 的 BPatch 对象设置为 main（）函数中的本地变量。或者，如果您需要使用 BPatch 作为全局变量，请在变异退出前手动分离所有 mutatee 进程。

这个程序接受进程 ID 和函数名称作为命令行参数，然后在执行进程期间显示调用函数的总数量。您可以使用以下 Makefile 构建这两个文件：

```
DTS      = /opt/rh/devtoolset-11/root
CXXFLAGS = -g -I$(DTS)/usr/include/dyninst
LBITS   := $(shell getconf LONG_BIT)

ifeq ($(LBITS),64)
  DYNINSTLIBS = $(DTS)/usr/lib64/dyninst
else
  DYNINSTLIBS = $(DTS)/usr/lib/dyninst
endif

.PHONY: all
all: count exercise

count: count.C
 g++ $(CXXFLAGS) count.C -I /usr/include/dyninst -c
 g++ $(CXXFLAGS) count.o -L $(DYNINSTLIBS) -ldyninstAPI -o count

exercise: exercise.C
 g++ $(CXXFLAGS) exercise.C -o exercise

.PHONY: clean
clean:
 rm -rf *~ *.o count exercise
```

要使用 Red Hat Developer Toolset 的 g++ 编译器在命令行中编译两个程序，请运行 make 工具程序：

```
$ scl enable devtoolset-11 make
```

```
g++ -g -I/opt/rh/devtoolset-11/root/usr/include/dyninst count.C -c
g++ -g -I/opt/rh/devtoolset-11/root/usr/include/dyninst count.o -L /opt/rh/devtoolset-
11/root/usr/lib64/dyninst -ldyninstAPI -o count
g++ -g -I/opt/rh/devtoolset-11/root/usr/include/dyninst exercise.C -o exercise
```

这会创建一个名为 exercise 的新二进制文件，并在当前工作目录中 计数。

在一个 shell 会话中，按如下所示执行 练习 二进制文件，并等待它提示您进入起始数字：

```
$ ./exercise
Enter the starting number:
```

不要输入这个数字。相反，启动另一个 shell 会话并在提示符下键入以下内容来设置 DYNINSTAPI_RT_LIB 环境变量并执行 计数 二进制文件：

```
$ export DYNINSTAPI_RT_LIB=/opt/rh/devtoolset-
11/root/usr/lib64/dyninst/libdyninstAPI_RT.so
$ ./count `pidof exercise` print_iteration
Finding function print_iteration(): OK
Instrumenting function print_iteration(): OK
Waiting for process 8607 to exit...
```

现在，切换回第一个 shell 会话，并根据 练习 程序请求的起始数字输入。例如：

```
Enter the starting number: 5
Iteration number 5
Iteration number 4
Iteration number 3
Iteration number 2
Iteration number 1
```

当 练习 程序终止时，计数 程序会显示在执行 print_iteration（） 函数的次数：

```
Function executed 5 times.
```

15.3. 其它资源

有关 Dyninst 及其功能的更多信息，请参阅以下列出的资源。

安装的文档

devtoolset-11-dyninst-doc 软件包会在 /opt/rh/devtoolset-11/root/usr/share/doc/devtoolset-11-dyninst-doc-11.0.0/ 目录中安装以下文档：

- *Dyninst Programmer 的指南 - Dyninst API 的详细描述存储在 DyninstAPI.pdf 文件中。*
- *DynC API Programmer 的指南 - DynC API 简介存储在 dynC_API.pdf 文件中。*
- *ParseAPI Programmer 的指南 - ParseAPI 简介并存储在 ParseAPI.pdf 文件中。*
- *PatchAPI 程序员指南 - 补丁API简介存储在 PatchAPI.pdf 文件中。*
- *ProcControlAPI Programmer 的指南 - ProcControlAPI.pdf 文件中存储了对 ProcControlAPI.pdf 的详细描述。*
- *StackwalkerAPI Programmer 的指南 - StackwalkerAPI 的详细描述存储在 stackwalker.pdf 文件中。*
- *SymtabAPI Programmer 的指南 - SymtabAPI 简介存储在 SymtabAPI.pdf 文件中。*
- *说明 API 参考手册 - 指令API的详细描述保存在 instructions API.pdf 文件中。*

有关如何在系统中安装这个软件包的详情请参考 第 15.1 节 “安装 Dyninst”。

在线文档

- *Dyninst Home Page - 项目主页提供了其他文档和相关发布的链接。*
- *Red Hat Enterprise Linux 7 SystemTap 发起者指南 - 用于 Red Hat Enterprise Linux 7 的 SystemTap 入门指南，提供 SystemTap 及其用法简介。*
- *Red Hat Enterprise Linux 7 SystemTap Tapset Reference - Red Hat Enterprise Linux 7 的 SystemTap Tapset 参考 提供了有关 SystemTap 的进一步详情。*

另请参阅

- *第 1 章 Red Hat Developer Toolset - Red Hat Developer Toolset 概述以及如何在您的系统中安装它的更多信息。*

- *第 12 章 SystemTap - SystemTap 简介以及有关如何使用它监控正在运行的系统活动的说明。*

- *第 13 章 Valgrind - 使用 Valgrind 工具分析应用程序并检测内存错误和内存管理问题的说明，如使用未初始化内存、分配和释放内存，以及在系统调用中使用不正确的参数。*

- *第 14 章 OProfile - 使用 OProfile 工具确定哪个部分代码消耗最大 CPU 时间以及原因。*

部分 V. 编译器工具集

第 16 章 编译器工具集文档

三种编译器工具集的描述：

- LLVM Toolset
- Go Toolset
- Rust Toolset

已移到 Red Hat Developer Tools 下的单独文档集。

部分 VI. 获取帮助

第 17 章 访问红帽产品文档

红帽产品文档位于 https://access.redhat.com/site/documentation/，作为中央信息来源。它目前使用 23 种语言转换，并且为每个产品提供了从版本和技术说明不同的书籍到安装、用户和参考指南（HTML、PDF 和 EPUB 格式）。

以下是与本书直接或间接相关的文档列表。

Red Hat Developer Toolset

- Red Hat Developer Toolset 11.0 发行注记 - Red Hat Developer Toolset 11.0 发行注记。
- 使用 Red Hat Software Collections 容器镜像 - 使用 Red Hat Software Collections 容器镜像提供获取、配置和使用 Red Hat Software Collections 附带的容器镜像的说明，包括 Red Hat Developer Toolset 容器镜像。
- Red Hat Software Collections 打包指南 - Software Collections 打包指南 解释了 Software Collections 的概念，以及如何创建、构建和扩展它们。

Red Hat Enterprise Linux

- Red Hat Enterprise Linux 7 开发人员指南 - Red Hat Enterprise Linux 7 开发人员指南 提供有关库和运行时支持、编译和构建、调试和性能分析的更多信息。
- Red Hat Enterprise Linux 7 安装指南 - Red Hat Enterprise Linux 7 安装指南 解释了如何获取、安装和更新系统。
- Red Hat Enterprise Linux 7 系统管理员指南 - Red Hat Enterprise Linux 7 的 系统管理员指南 介绍了与 Red Hat Enterprise Linux 7 部署、配置和管理有关的信息。

第 18 章 联系全球支持服务

如果您有一个自助支持订阅，当红帽文档网站和客户门户网站无法提供您的问题的答案时，您可以联系全球支持服务 (GSS)。

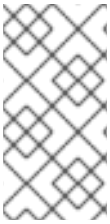
18.1. 收集所需信息

在联系 GSS 之前，应收集几项信息。

背景信息

在调用 GSS 前，请确保手动有以下背景信息：

- 运行产品的硬件类型、制作和型号
- 软件版本
- 最新升级
- 对系统进行的任何最近更改
- 问题说明和症状
- 有关此问题的任何信息或重要信息



注意

如果您忘记了 Red Hat 登录信息，可以在 https://access.redhat.com/site/help/LoginAssistance.html 中恢复。

诊断

另外，还需要 Red Hat Enterprise Linux 的诊断报告。此报告也称为 sosreport，用于创建报告的程序由 sos 软件包提供。要在您的系统中安装 sos 软件包及其所有依赖项：

```
# yum install sos
```

生成报告：

```
# sosreport
```

如需更多信息，请访问知识库文章 https://access.redhat.com/kb/docs/DOC-3593。

帐户和联系信息

为了帮助您，GSS 需要您的帐户信息来定制其支持，以及联系信息以返回给您。当您联系 GSS 可确保您：

- 红帽客户号或红帽网络(RHN)登录名称
- 公司名称
- 联系名称
- 首选联系方式（电话或电子邮件）以及联系信息（电话号码或电子邮件地址）

问题严重性

确定问题的严重性对于允许 GSS 团队确定其工作的优先级非常重要。有四个级别的严重性。

严重级别 1（紧急）

严重影响您用于生产目的的问题。它将停止您的业务操作，并且没有其他可绕过问题的解决方法。

严重级别 2（高）

软件可以正常运行的问题，但生产严重程度被降低。这会导致对业务运营造成大量影响，且不存在临时解决方案。

严重级别 3（中）

涉及软件部分非关键损失的问题。对您的业务有中等影响，业务通过使用临时解决方案来继续发挥作用。

严重级别 4（低）

一般使用问题，报告文档错误或将来产品改进的建议。

有关确定问题的严重性级别的更多信息，请参阅 https://access.redhat.com/support/policy/severity。

确定了问题严重性后，请通过客户门户网站中的" 连接 "选项或在 https://access.redhat.com/support/contact/technicalSupport.html 中提交服务请求。请注意，您需要您的红帽登录信息来提交服务请求。

如果严重程度为 1 或 2，则通过电话来跟踪您的服务请求。联系信息和营业时间可以通过 https://access.redhat.com/support/contact/technicalSupport.html 获得。

如果您有高级订阅，则 1 到 2 个情况会提供数小时支持。

高级订阅和标准订阅的周转率均可在 https://access.redhat.com/support/offerings/production/sla.html 找到。

18.2. 升级问题

如果您认为问题没有被正确处理，或被正确处理，您可以升级它。升级有两种类型：

技术升级

如果某个问题没有被适当解决，或者您需要一个更高级的资源来参加它。

管理升级

如果问题变得更为严重，或者您认为它要求具有更高的优先级。

有关升级（包括联系人）的更多信息，请访问 https://access.redhat.com/support/policy/mgt_escalation.html

18.3. 重新打开服务请求

如果与封闭的服务请求相关的信息（如问题重新发生），您可以通过红帽客户门户网站（

https://access.redhat.com/support/contact/technicalSupport.html 或调用本地支持中心）重新打开请求，请参阅 https://access.redhat.com/support/contact/technicalSupport.html。

重要

要重新打开服务请求，需要原始的 service-request 编号。

18.4. 其它资源

如需更多信息，请参阅下面列出的资源。

在线文档

- 入门 - 入门页面作为购买红帽订阅的人员的起点，并提供红帽欢迎套件和红帽支持快速指南以下载。
- 如何使用 RHEL 自助支持订阅？ - 为客户提供自助支持订阅的知识库文章。
- 红帽全球支持服务和公共邮件列表 - 知识库文章，回答有关公共邮件列表的频繁问题。

附录 A. 版本 11.0 的更改

以下小节包括了 Red Hat Developer Toolset 11.0 引入的功能及兼容性更改。该列表未满，将更新。

A.1. GCC 的更改

Red Hat Developer Toolset 11.0 由 GCC 11.2 提供。

自以前的 Red Hat Developer Toolset 发行版本以来，添加了或修改了以下功能：

常规改进

- GCC 现在默认为 DWARF 版本 5 调试格式。
- 诊断中显示的列号默认为实际列号并尊重多列字符。
- 直接线性代码向向量化时，会考虑整个功能。
- 如果每个条件表达式中包含比较同一变量，则可以将一系列条件表达式转换为 switch 语句。
- 流程优化改进：
  - 新的 IPA-modref pass 由 -fipa-modref 选项控制，跟踪函数调用的副作用并改进点分析的精度。
  - 通过 -fipa-icf 选项控制相同的代码是明显的，以增加统一功能的数量并减少编译内存使用。
- 链路优化改进：
  - 改进了连接过程中的内存分配，以减少峰值内存用量。

- 在 IDE 中使用新的 GCC_EXTRA_DIAGNOSTIC_OUTPUT 环境变量，您可以在不调整构建标志的情况下请求机器可读的"fix-it hints"。

- 由 -fanalyzer 选项运行的静态分析器会显著提高，提供很多程序错误修复和增强。

- 为了缓解 CVE-2021-42574，在 GCC 中添加了一个新的警告，并带有 RHSA-2021:4669 公告的发行版本。这个新的 -Wbiway=[none|unpaired|any] 警告可能存在危险(BiDi) Unicode 字符并有三个级别：

  - -Wbiway=unpaired （默认）警告错误终止 BiDi 上下文。

  - -Wbiway=none 关闭警告。

  - -Wbiway=any 警告任何使用 BiDi 字符。

特定于语言的改进

C family

- C 和 C++ 编译器支持 OpenMP 构造和 OpenMP 5.0 规范的"非恢复循环"例程。

- 属性：

  - 新的 no_stack_protector 属性标记不应通过堆栈保护来检测的功能(-fstack-protector)。

  - 改进的 malloc 属性可用于识别分配器和放大器 API 对。

- 新警告：

  - -Wsizeof-array-div 通过 -Wall 选项启用，它会警告两个 sizeof 操作符的不同（当第一个应用到一个数组，而 divisor 与数组的大小不同时）。

- -Wstringop-overread 默认启用，警告字符串对字符串功能的调用，尝试将以参数形式传递给它们的阵列结束。

- 增强的警告：

  - -Wfree-nonheap-object 检测到更多调用实例来利用没有从动态内存分配功能返回的指针来取消分配函数。

  - -Wmaybe-uninitialized 诊断将指针和引用未初始化的内存传递给使用 const-qualified 参数的功能。

  - -Wuninitialized 检测到从未初始化动态分配的内存中读取。

C

- 通过 -std=c2x 和 -std=gnu2x 选项支持来自即将发布的 C2X 版本中的几个新功能。例如：

  - 标准属性被支持。

  - 支持 __has_c_attribute preprocessor operator。

  - 标签可能会在 compound 语句的末尾出现。

C++

- 默认模式改为 -std=gnu++17。

- 现在，C++ 库 libstdc++ 改进了 C++17 支持。

- 实施了几个新的 C++20 功能。请注意，C++20 支持是实验性的。

  有关功能的更多信息，请参阅 C++20 语言功能。

- C++ 前端对一些即将出现的 C++23 草案功能提供了实验性支持。

- 新警告：

  - -Wctad-maybe-unsupported（默认禁用），提醒在类型为 deduction 指南中的执行类模板参数。

  - -Wrange-loop-construct （由 -Wall 启用），当基于范围循环时警告是创建不必要的且资源低效的副本。

  - -Wmismatched-new-delete （由 -Wall 启用），警告对 Operator 的调用，并发出从不匹配形式的 Operator 返回的指针，或从其他不匹配分配功能返回。

  - -Wvexing-parse （默认启用）会警告大多数 vexing 解析规则：当声明与变量定义类似时，C++ 语言需要被解释为函数声明。

特定于架构的改进

64 位 ARM 架构

- Armv8-R 架构通过 -march=armv8-r 选项进行支持。

- GCC 可自动执行增加、减法、乘法、乘法和减去复杂数字上的操作。

AMD 和 Intel 64 位构架

- 支持以下 Intel CPU：Sapphire Rapids、Alder Lake 和 Rocket Lake。

-

添加了对 Intel AVX-VNNI 的新 ISA 扩展支持。-mavxvnni 编译器切换控制 AVX-VNNI 内部。

- 新的 -march=znver3 选项支持基于 znver3 内核的 AMD CPU。
- the x86-64 psABI supplement 中定义的三个微架构级别通过新的 -march=x86-64-v2、-march=x86-64-v3 和 -march=x86-64-v4 选项支持。

A.2. BINUTILS 的更改

Red Hat Developer Toolset 11.0 与 binutils 2.36 一起发布。

自以前的 Red Hat Developer Toolset 发行版本以来，添加了或修改了以下功能：

assembler

- 在 Intel 构架中，支持 AMX、AVX VNI、HRESET、key Locker、TDX 和 UINTR 指令。
- 当设置 ELF 部分的 link order 属性时，您可以使用数字部分索引而不是符号名称。
- 以下 ARM 内核受支持：Cortex-A78、Cortex-A78AE、Cortex-A78C、Cortex-X1、Cortex-R82、Neoverse V1 和 Neoverse N2。
- 在 64 位 ARM 架构中，支持 Armv8-R 和 Armv8.7-A ISA 扩展。
- 添加了 .nop 指令，它会生成一个可在任何目标上运行的单个 no-operation 指令。
- 支持 SHF_GNU_RETAIN 标志。它指定这个部分不应被链路器收集。这个标志可以被应用到 .section 指令中的 R 标志。

linker

- 添加了一个新的 libdep 插件。它记录链接静态库中的依赖项，并在执行最终链接时使用它们。

- 添加了一个新的 --error-handling-script=<NAME > 命令行选项。当遇到未定义符号或缺少库时，它会运行一个帮助程序脚本。

- 现在，链接器会在 .ctf 部分中分离类型。您可以使用新的 --ctf-share-types 命令行选项来指定链接者如何进行此操作。这个选项的默认值是 shared -unconflicted，生成最紧凑的输出。

- 默认情况下，linker 省略 .ctf 部分中的变量部分，节省空间。对于自己模拟符号表的项目来说，这种行为可能不相同，它们没有反映在 ELF 符号表中。

- 支持 SHF_GNU_RETAIN ELF 部分标记。此标志指定部分不应被 linker 收集的垃圾回收。

其他二进制工具

- nm: 添加了一个新的命令行选项 --ifunc-chars=CHARS，用于指定一个或两个字符的字符串。显示全局 ifunc 符号时，第一个字符用作类型字符。如果显示本地 ifunc 符号，则使用第二个字符。

- ar ：之前未使用的 I 修饰符可用于指定静态库的依赖项。此 I 选项的参数（或其较长的格式为 --record-libdeps）保存在存档的 __.LIBDEP 成员中，其链接者可以在链接时读取。

- readelf: 使用 --lto-syms 命令行选项，可以显示 LTO 符号表部分的内容。

- readelf 接受 -C 命令行选项，该选项启用符号名称。另外，增加了 --demangle=<style >、--no-demangle、--recurse-limit 和 --no-recurse-limit 选项。

- 为了缓解 CVE-2021-42574，在 binutils 中添加了一个新命令行选项，其中包含 RHSA-2021:4730 公告。

  显示名称或字符串（readelf、String、nn 和 objdump）的工具现在有一个新的 --unicode (-U)命令行选项，用于控制 Unicode 字符的处理方式。可以为选项设置以下值：

  - --Unicode=default 可正常对待 BiDi 字符。当 --unicode 选项没有被使用时，这是默认的 behaviour。

- --Unicode=locale 会根据当前的区域设置显示 BiDi 字符。
- --Unicode=hex 显示为十六进制值。
- --Unicode=escape 将 BiDi 字符显示为 Unicode 转义序列。
- --Unicode=highlight 会显示 BiDi 字符，如果输出设备支持，则 Unicode 转义序列以红色突出显示。

A.3. ELFUTILS 的更改

Red Hat Developer Toolset 11.0 带有 elfutils 0.185。

自以前的 Red Hat Developer Toolset 发行版本以来，添加了或修改了以下功能：

- eu-elflint 和 eu-readelf 工具现在可以识别并显示 ELF _GNU_RETAIN 和 SHT_X86_64_UNWIND 标记。
- DEBUGINFOD_SONAME 宏已添加到 debuginfod.h 中。此宏可与 dlopen 功能一起使用，从而从应用程序动态加载 libdebuginfod.so 库。
- 在 debuginfod-client 库中添加了一个新的 debuginfod_set_verbose_fd。此功能通过将详细输出重定向到单独的文件来增强 debuginfod_find_* 查询功能。
- 设置 DEBUGINFOD_VERBOSE 环境变量现在会显示有关哪些服务器 连接到哪个 服务器以及这些服务器的 HTTP 响应的更多信息。
- debuginfod 服务器提供了一个新的 thread-busy 指标和更详细的错误指标，以便更轻松地检查 debuginfod 服务器上运行的进程。
- libdw 库透明地处理 DW_FORM_indirect 位置值，以便 dwarf_whatform 功能返回属性的实际 FORM。

- 要减少网络流量，debuginfod-client 库会将负结果存储在缓存中，客户端对象可以重复使用现有的连接。

A.4. DWZ 的更改

Red Hat Developer Toolset 11.0 带有 dwz 0.14。

自以前的 Red Hat Developer Toolset 发行版本以来，添加了或修改了以下功能：

- 支持 DWARF 版本 5 调试格式。
- DWARF 补充对象文件可使用 .debug_sup 部分生成。
- 添加了一个新的实验性优化功能，利用了 C++ 中的一个定义规则。
- 支持 DW_OP_GNU_variable_value 表达式 opcode。
- 已修复大量程序错误，并改进了性能。

A.5. GDB 中的更改

Red Hat Developer Toolset 11.0 与 GDB 10.2 一起发布。

自以前的 Red Hat Developer Toolset 发行版本以来，添加了或修改了以下功能：

新功能

- 在支持此功能的架构中默认启用多线程符号载入。此更改为带有很多符号的程序提供更好的性能。
- 文本用户界面(TUI)窗口可以水平排列。

- GDB 支持同时调试多个目标连接，但这个支持是实验性并有限。例如，您可以将每个人连接到在不同的机器上运行的不同远程服务器，也可以使用一个不合者调试本地原生进程或核心转储或某些其他进程。

新的和改进的命令

- 新的 tui new-layout name window weight [window weight...] 命令创建一个新的文本用户界面(TUI)布局，您还可以指定布局名称和显示窗口。

- 改进的 别名 [-a] [--] alias = command [default-args] 命令在创建新别名时指定默认参数。

- set exec-file-mismatch 和 show exec-file-mismatch 命令集和显示新的 exec-file-mismatch 选项。当 GDB 附加到正在运行的进程时，这个选项控制在当前由 GDB 加载的可执行文件和用于启动进程的可执行文件之间是否检测到不匹配时 GDB 如何做出反应。

Python API

- gdb.register_window_type 函数在 Python 中实现新的 TUI 窗口。

- 现在，您可以查询动态类型。gdb.Type 类的实例可以具有新的布尔值属性 动态，而 gdb.Type.sizeof 属性的值可以具有 None 值的动态类型。如果 Type.fields（） 返回了动态类型的字段，则其 bitpos 属性的值可以是 None。

- 新的 gdb.COMMAND_TUI constant 将 Python 命令注册为 TUI 帮助类命令的成员。

- 新的 gdb.PendingFrame.architecture（） 方法检索待处理帧的架构。

- 新的 gdb.Architecture.registers 方法会返回 gdb.RegisterDescriptorIterator 对象，它会返回 gdb.RegisterDescriptor 对象。此类对象不提供注册的值，但有助于了解哪些寄存器可用于架构。

- 新的 gdb.Architecture.register_groups 方法会返回 gdb.RegisterGroupIterator 对象，它会返回 gdb.RegisterGroup 对象。这样的对象有助于了解哪些寄存器组可用于架构。

A.6. LTRACE 的更改

Red Hat Developer Toolset 11.0 带有 ltrace 0.7.91。

从以前的 Red Hat Developer Toolset 版本开始对以下功能进行了修改：

- 如果在 $XDG_CONFIG_DIRS 补丁文件中指定了路径，则不会给出诊断。

A.7. STRACE 的更改

Red Hat Developer Toolset 11.0 带有 strace 5.13。

自以前的 Red Hat Developer Toolset 发行版本以来，添加了或修改了以下功能：

行为更改

- 修改后的 % 进程 类包含与进程生命周期（计算、执行和终止）关联的系统调用：
  - 新调用： kill、tkill、tgkill、pidfd_send_signal、rt_sigqueueinfo
  - 删除了调用： arch_prctl 和 unshare

改进

- 新的 -n (--syscall-number)选项打印系统调用号。
- 新的 --secontext[=full] 选项会显示 SELinux 上下文。
- 系统将实施 poke 注入，并添加了两个新选项：-- inject=SET:poke_enter= 和 --inject=SET:poke_exit=。
- 在 IBM POWER 构架中，添加了系统调用向量(SCV)ABI 支持。

- *libdw- 为非原生分配启用基于堆栈的追踪。*

- *netlink 数据以更结构化的方式打印。*

- *以下系统调用的解码已实现：close_range、epoll_pwait2、faccessat2、landlock_add_rule、landlock_create_ruleset、landlock_restrict_self、mount_setattr 和 process_madvise。*

- *对以下系统调用的解码已被改进：io_uring_setup、membarrier、perf_event_open、pidfd_open。*

- *实现了 GPIO_* 和 TEE_* ioctl 命令的解码。*

- *以下 ioctl 命令的解码已实现： fs_IOC_FS[gs]etXATTR, fs_IOC_[gs]etFLAGS, fs_IOC32_[gs]etFLAGS, IOOP_CONFIGURE, sIOCADDMULTI,sIOCDMULTI, SIOCGIFENCAP,SIOCOUTQNSD,SIOCSIFENCAP,SIOCSIFHWBROADCAST,UBI_IOCRPEB 和 UBI_IOCSPEB、 V4L2BUF_TYPE_META_CAPTURE, V4L2_BUF_TYPE_META_OUTPUT 和 VIDIOC_QUERY_EXT_CTRL.*

- *实施了 NT_PRSTATUS 和 NT_FPREGSET regsets，并实现了 Pabrt_GETREGSET 和 Pabrt_SETREGSET ptrace 请求。*

- *以下 ptrace 请求的 regs 参数已被实施： PDIB_GETREGS、PDIB_GETREGS64、PSYS_SETREGS、PDIB_SETREGS64、 PSYS_GETFPREGS 和 P Wright_SETFPREGS。*

- *实施了 IPC_INFO 和 MSG_INFO msgctl 系统调用命令的 struct msginfo 参数。*

- *实现了 MSG_STAT 和 MSG_STAT _ ANY msgctl 系统调用命令的 struct msqid_ ds 参数的解码。*

- *实施了 IPC_INFO 和 SEM_INFO semctl 系统调用命令的 struct seminfo 参数的解码。*

- *实施了 IPC_SET、IPC_STAT、SEM_STAT 和 SEM_STAT _ANY semctl 系统调用参数的 struct semid_ ds 参数的解码。*

- 已实现 IPC_INFO shmctl 系统调用命令 struct shminfo 参数的解码器。

- 实现了 SHM_INFO shmctl 系统调用命令 struct shm_info 参数的解码。

- 实现了 SHM_STAT 和 SHM_STAT _ANY shmctl 系统调用命令的 struct shmid_ ds 参数的解码。

- IFLA_BRPORT_* netlink 属性的解码已更新，以匹配 Linux 5.12 内核。

- 以下常量列表已更新：*_magic, aLG_* , audit_*, bpf_*, btr FS_*, cap _*, close _RANGE_*, dev CONF_*, eth_*, fAN _*, fAN_* iflA_*, iNET_DIAG_*, ior ING_*, ipv 6_* , ip_* , k EXEC_*, key CTL_* , key_* , kvm_* , kvm_*, lOOP_*, mdbA_* , memBARRIER_CMD_*, mPOL_*, ms_* , mtD_* , nda_* , nFT_MSG_* nlMSGERR_*, nt_*, pr _*, ptp _PEROUT_*, p eligible_* , res OLVE_*, r TAX_*, rt A_* , rtc_*, rtc_*, rtc_*, rtm_*, rtNH_*, rtPROT_*, sctp_*, se GV_*, so _*, stat X_* , st_* , sys_*, tca_* , tca_* TRAP_*、UFFDIO_*、UFFD_*、UFFD_* 和 V4L2_*。

- ioctl 命令列表已更新，以匹配 Linux 5.13 内核更新中的这些列表。

- 随着 RHEA-2022:4635 公告的发布，strace 现在可在实际 SELinux 上下文和从 SELinux 上下文数据库中提取的定义之间显示不匹配。

  使用 不匹配 参数扩展 strace 的现有 --secontext 选项。这个参数可让您只打印预期的上下文以及实际不匹配的上下文。输出用双感叹号(!!)分隔，第一个是实际上下文，然后是预期上下文。在下面的示例中，full,mismatch 参数打印预期的完整上下文以及实际的上下文，因为上下文的用户部分不匹配。但是，在使用单独的 mismatch 时，它只检查上下文的类型部分。预期的上下文不会打印，因为上下文的类型部分匹配。

  ```
  [...]
  $ strace --secontext=full,mismatch -e statx stat /home/user/file
  statx(AT_FDCWD, "/home/user/file"
  [system_u:object_r:user_home_t:s0!!unconfined_u:object_r:user_home_t:s0], ...

  $ strace --secontext=mismatch -e statx stat /home/user/file
  statx(AT_FDCWD, "/home/user/file" [user_home_t:s0], ...
  ```

  SELinux 上下文不匹配通常会导致与 SELinux 相关的访问控制问题。系统调用 traces 中打印的不匹配可显著加快 SELinux 上下文正确性的检查。系统调用 traces 也可以解释有关访问控

制检查的特定内核行为。

程序错误修复

- 修正了 SIOCGIFINDEX、SIOCBRADDIF 和 SIOCBR DIF ioctl 命令的解码。
- clock_gettime64、clock_settime64、clock_adjtime64 和 lock_getres_time64 系统调用添加到 %clock trace 类。
- statx 系统调用添加到 %fstat 跟踪类中。
- 在以前的版本中，strace 在网络接口名称打印中使用不足的缓冲区大小。这导致在试图显示需要引用的打印接口名称时导致断言，例如： -xx 模式中的名称大于 4 个字符。随着 RHEA-2022:4635 公告的发布，这个程序错误已被解决。

A.8. SYSTEMTAP 的更改

Red Hat Developer Toolset 11.0 与 SystemTap 4.5 一起发布。

自以前的 Red Hat Developer Toolset 发行版本以来，添加了或修改了以下功能：

- 32 位浮动点变量会自动设置为双引号，因此可以作为 $context 变量直接访问。
- enum 值可作为 $context 变量来访问。
- BPF uconversions tapset 已被扩展，包含更多 tapset 功能来访问用户空间中的值，如 user_long_error（）。
- 改进了并发控制，以便在大型服务器上提供稳定的操作。

有关显著变化的详情，请查看上游 SystemTap 4.5 发行注记。

A.9. VALGRIND 的变化

Red Hat Developer Toolset 11.0 与 Valgrind 3.17.0 一起发布。

自以前的 Red Hat Developer Toolset 发行版本以来，添加了或修改了以下功能：

- Valgrind 可以读取 DWARF 版本 5 调试格式。
- Valgrind 支持对 debuginfod 服务器的调试查询。
- ARMv8.2 处理器指令部分被支持。
- POWER10 处理器上的 Power ISA v.3.1 指令部分被支持。
- 支持 IBM z14 处理器指令。
- 大多数 IBM z15 指令都被支持。Valgrind 工具套件支持 miscellaneous-instruction-extensions 工具 3，为 IBM z15 处理器支持向量化工具 2。因此，Valgrind 运行使用 GCC -march=z15 编译的程序，并改进了性能和调试体验。
- --track-fds=yes 选项遵循 -q (-quiet)，并默认忽略标准文件描述符 stdin、stdout 和 stderr。要跟踪标准文件描述符，请使用 --track-fds=all 选项。
- DHAT 工具有两个新的操作模式：-- mode=copy 和 --mode=ad-hoc。

A.10. DYNINST 的更改

Red Hat Developer Toolset 11.0 带有 Dyninst 11.0.0。

自 Red Hat Developer Toolset 11.0 之前的发行版本起添加了以下功能：

- 支持 debuginfod 服务器并获取单独的 debuginfo 文件。
- 改进了对流程链路表(PLT)存根的间接调用的检测。
- 改进了 C++ 名称 demangling。
- 修复了代码发出过程中内存泄漏的问题。

A.11. ANNOBIN 的更改

Red Hat Developer Toolset 11.0 带有 Annobin 9.82。

自以前的 Red Hat Developer Toolset 发行版本以来，添加了或修改了以下功能：

GCC 插件

- 支持 ARM 和 RISCV 目标。
- 支持 LTO 编译器。

Annocheck

- 在 verbose 模式中，报告跳过特定测试的原因。
- 某些消息以颜色突出显示。
- 添加了一些 GO 测试。
- 在 64 位 ARM 架构中，增加了对 BTI 和 PAC 安全功能的测试。
- 为了缓解 CVE-2021-42574，添加了一个新的测试来检测符号名称中存在多个字节字符。随着 RHSA-2021:4729 公告的发行版本，已在 Annobin 中实现这个更改。