Oracle® Solaris Studio 12.3 代码分析器用户指南

文件号码 E26467
2011 年 12 月

ORACLE

120220@25097

目录

前言

《Oracle Solaris Studio 12.3 代码分析器用户指南》介绍了如何使用代码分析器工具，其中包括如何使用编译器、Discover 和 Uncover 收集静态数据、动态内存数据和代码覆盖数据，以及如何运行代码分析器 GUI 分析和显示数据。

受支持的平台

此 Oracle Solaris Studio 发行版支持使用以下体系结构的平台：运行 Oracle Solaris 操作系统的 SPARC 系列处理器体系结构，以及运行 Oracle Solaris 或特定 Linux 系统的 x86 系列处理器体系结构。

本文档使用以下术语说明 x86 平台之间的区别：

- "x86" 泛指 64 位和 32 位的 x86 兼容产品系列。
- "x64" 指特定的 64 位 x86 兼容 CPU。
- “32 位 x86”指出了有关基于 x86 的系统的特定 32 位信息。

在 SPARC 和 x86 系统中，特定于 Linux 系统的信息仅指受支持的 Linux x86 平台，而特定于 Oracle Solaris 系统的信息仅指受支持的 Oracle Solaris 平台。

有关受支持的硬件平台和操作系统发行版的完整列表，请参见《Oracle Solaris Studio 12.3 发行说明》。

Oracle Solaris Studio 文档

可以查找 Oracle Solaris Studio 软件的完整文档，如下所述：

- 产品文档位于 Oracle Solaris Studio 文档 Web 站点，包括发行说明、参考手册、用户指南和教程。
- 代码分析器、性能分析器、线程分析器、dbxtool、DLight 和 IDE 的联机帮助可以在这些工具中通过 "Help"（帮助）菜单以及 F1 键和许多窗口和对话框上的 "Help"（帮助）按钮获取。
- 命令行工具的手册页介绍了工具的命令选项。

相关的第三方 Web 站点引用

本文档引用了第三方 URL，以用于提供其他相关信息。

注 – Oracle 对本文档中提到的第三方 Web 站点的可用性不承担任何责任。对于此类站点或资源中的（或通过它们获得的）任何内容、广告、产品或其他资料，Oracle 并不表示认可，也不承担任何责任。对于因使用或依靠此类站点或资源中的（或通过它们获得的）任何内容、产品或服务而造成的或连带产生的实际或名义损坏或损失，Oracle 概不负责，也不承担任何责任。

开发者资源

对于使用 Oracle Solaris Studio 的开发者，可访问 Oracle 技术网 Web 站点来查找以下资源：

- 有关编程技术和最佳做法的文章
- 软件最新发行版完整文档的链接
- 有关支持级别的信息
- 用户论坛。

获取 Oracle 支持

Oracle 客户可通过 My Oracle Support 获取电子支持。有关信息，请访问 `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info`，或访问 `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs`（如果您听力受损）。

印刷约定

下表介绍了本书中的印刷约定。

表 P–1 印刷约定

字体或符号	含义	示例
`AaBbCc123`	命令、文件和目录的名称；计算机屏幕输出	编辑 `.login` 文件。 使用 `ls -a` 列出所有文件。 `machine_name% you have mail.`
`AaBbCc123`	用户键入的内容，与计算机屏幕输出的显示不同	`machine_name%` **`su`** `Password:`

表 P–1 印刷约定 （续）

字体或符号	含义	示例
aabbcc123	要使用实名或值替换的命令行占位符	删除文件的命令为 `rm` *filename*。
AaBbCc123	保留未译的新词或术语以及要强调的词	这些称为 *Class* 选项。 **注意**：有些强调的项目在联机时以粗体显示。
新词术语强调	新词或术语以及要强调的词	**高速缓存**是存储在本地的副本。 请**勿**保存文件。
《书名》	书名	阅读《用户指南》的第 6 章。

命令中的 shell 提示符示例

下表显示了 Oracle Solaris OS 中包含的缺省 UNIX shell 系统提示符和超级用户提示符。请注意，在命令示例中显示的缺省系统提示符可能会有所不同，具体取决于 Oracle Solaris 发行版。

表 P–2 shell 提示符

shell	提示符
Bash shell、Korn shell 和 Bourne shell	`$`
Bash shell、Korn shell 和 Bourne shell 超级用户	`#`
C shell	`machine_name%`
C shell 超级用户	`machine_name#`

◆ ◆ ◆ 第 1 章

1 简介

Oracle Solaris Studio 代码分析器是一套集成的工具，用于帮助 C 和 C++ 应用程序开发者针对 Oracle Solaris 开发出既安全又强健的高质量软件。

本章包括有关下列内容的信息：

代码分析器分析的数据

代码分析器可以分析以下三种类型的数据：

- 编译期间检测到的静态代码错误
- 由内存错误搜索工具 Discover 检测到的动态内存访问错误和警告
- 由代码覆盖工具 Uncover 测量的代码覆盖数据

除了允许您访问各种分析类型以外，代码分析器还集成了静态代码检查和动态内存访问检查，用于向在代码中发现的错误添加置信度级别。通过将静态代码检查与动态内存访问分析和代码覆盖分析结合使用，将可以在应用程序中找到其他错误检测工具通过自身无法找到的许多重要错误。

代码分析器还可精确定位代码中的核心问题，修复这些问题后可能会消除其他问题。一个核心问题通常会伴有多个其他问题，例如，因为这些问题具有一个通用分配点或出现在同一函数中的同一数据地址上。

静态代码检查

静态代码检查可在编译期间检测代码中出现的常见编程错误。C 和 C++ 编译器的 `-xanalyze=code` 选项利用编译器既广泛又成熟的控制和数据流分析框架来分析应用程序是否存在潜在的编程和安全缺陷。

有关收集静态错误数据的信息，请参见第 13 页中的 “收集静态错误数据”。

有关代码分析器分析的静态代码错误的列表，请参见第 19 页中的 “静态代码问题”。

动态内存访问检查

通常，很难以找到代码中与内存相关的错误。在运行程序之前使用 Discover 对程序进行检测时，Discover 会在程序执行期间动态捕捉并报告内存访问错误。例如，如果您的程序分配了一个数组但未将其初始化，然后尝试从数组中的某个位置执行读取操作，程序可能会出现异常行为。如果使用 Discover 对程序进行检测，然后运行此程序，Discover 将捕捉该错误。

有关收集动态内存访问错误数据的信息，请参见第 14 页中的 “收集动态内存访问数据”。

有关代码分析器分析的动态内存访问问题的列表，请参见第 20 页中的 “动态内存访问问题”。

代码覆盖检查

代码覆盖是软件测试的重要部分。该工具提供了测试时执行或未执行的代码区域的相关信息，使您可以改进测试套件以测试更多代码。代码分析器使用由 Uncover 收集的数据确定程序中哪些函数是未覆盖函数，以及在添加覆盖相关函数的测试时将添加到应用程序总覆盖的覆盖百分比。

有关收集代码覆盖数据的信息，请参见第 15 页中的 “收集代码覆盖数据”。

代码分析器的使用要求

代码分析器处理从使用 Oracle Solaris Studio 12.3 C 或 C++ 编译器编译的二进制文件中收集的静态错误数据、动态内存访问错误数据以及代码覆盖数据。

代码分析器在运行 Solaris 10 10/08 操作系统（或更高的 Solaris 10 Update 版本）或 Oracle Solaris 11 的基于 SPARC 或基于 x86 的系统上运行。

代码分析器 GUI

使用编译器、Discover 或 Uncover 收集数据之后，可以启动代码分析器 GUI 来显示并分析问题。

对于每个问题，代码分析器显示问题描述、发现问题的源文件的路径名以及该文件中突出显示相关源代码行的代码片段。

在代码分析器中，可以执行以下操作：

- 显示问题的更多详细信息。对于静态问题，详细信息包括“错误路径”。对于动态内存访问问题，详细信息包括“调用堆栈”，如果数据可用，还包括“分配堆栈”和“可用堆栈”。
- 打开发现问题的源文件。
- 从“错误路径”或堆栈中的函数调用跳转到关联的源代码行。
- 查找函数在程序中的所有使用情况。
- 跳转到函数声明。
- 跳转到被覆盖函数或覆盖函数的声明。
- 显示函数的调用图。
- 显示关于每个问题类型的更多信息，包括代码示例和可能的原因。
- 按分析类型、问题类型和源文件过滤显示的问题。
- 隐藏已查看的问题并关闭不需要关注的问题。

有关使用 GUI 的详细信息，请参见 GUI 中的联机帮助以及《Oracle Solaris Studio 12.3 代码分析器教程》。

快速入门

下面是一个示例，该示例可实现以下功能：编译程序以收集静态代码数据、使用调试信息重新进行编译、使用 Discover 对程序进行检测并运行此程序以收集动态内存访问数据、使用 Uncover 对程序进行检测以收集代码覆盖数据，然后启动代码分析器来显示所收集的数据。

```
% cc -xanalyze=code *.c
% cc -g *.c
% cp a.out a.out.save
% discover -a a.out
% a.out
% cp a.out.save a.out
% uncover a.out
% a.out
% uncover -a a.out.uc
% code-analyzer a.out
```

◆ ◆ ◆ 第 2 章

收集数据和启动代码分析器

所收集的供代码分析器进行分析的数据存储在包含源代码文件的目录的 *binary_name*.analyze 目录中。*binary_name*.analyze 目录由编译器、Discover 或 Uncover 创建（具体取决于在收集程序的数据时最先运行这些工具中的哪一个）。

本章包括有关下列内容的信息：

收集静态错误数据

要收集 C 或 C++ 程序的静态错误数据，请在 Oracle Solaris Studio 12.3 C 或 C++ 编译器中使用 -xanalyze=code 选项来编译程序。（在先前的 Oracle Solaris Studio 发行版中，-xanalyze=code 选项在编译器中不可用。）使用此选项时，编译器会自动提取静态错误并将数据写入 *binary_name*.analyze 目录的 static 子目录中。

如果使用 -xanalyze=code 选项编译程序，然后在单独的步骤中链接此程序，您还需要在链接步骤中包括 -xanalyze=code 选项。

编译器并不能检测代码中的所有静态错误。

- 某些错误依赖于仅在运行时可用的数据。例如，假设给定以下代码，编译器不会检测 ABW（beyond Array Bounds Write，数组越界写）错误，因为它无法检测从某个文件读取的 ix 值是否位于 [0,9] 范围之外：

```
void f(int fd, int array[10])
{
  int ix;
  read(fd, &ix, sizeof(ix));
  array[ix] = 0;
}
```

- 有些错误并不明确，换句话说，它们在代码中可能是实际存在的错误，但也可能不是。编译器不报告这些错误。
- 在此发行版中，编译器不检测某些复杂的错误。

收集静态错误数据之后，您可以启动代码分析器 GUI 来分析和显示数据（请参见第 16 页中的 “启动代码分析器 GUI”），或重新编译程序以便可以收集动态内存访问数据或代码覆盖数据。

收集动态内存访问数据

收集 C 或 C++ 程序的动态内存访问数据的过程包含以下两个步骤：使用 Discover 检测二进制文件，然后运行检测过的二进制文件。

要使用 Discover 检测程序以收集数据供代码分析器使用，必须已使用 Oracle Solaris Studio 12.3 C 或 C++ 编译器对程序进行了编译。使用 `-g` 选项进行编译可生成调试信息，从而使代码分析器可以显示动态内存访问错误和警告的源代码和行号信息。

如果在不进行优化的情况下编译程序，Discover 将提供源代码级别的最完整内存错误检测。如果编译时进行优化，将检测不到某些内存错误。

有关 Discover 能够检测或不能检测的特定类型二进制文件的信息，请参见《Oracle Solaris Studio 12.3： Discover 和 Uncover 用户指南》中的“可以使用重新定义标准内存分配函数的二进制文件”和《Oracle Solaris Studio 12.3： Discover 和 Uncover 用户指南》中的“不能使用使用预装或审计的二进制文件”。

注 – 您可以生成程序一次来同时用于 Discover 和 Uncover。但是，您不能检测已检测过的二进制文件，因此，如果还打算使用 Uncover 收集覆盖数据，请在使用 Discover 进行检测之前先保存此二进制文件的副本以避免出现此问题。例如：

```
cp a.out a.out.save
```

从二进制文件中收集动态内存访问数据：

1. 使用 Discover 并结合 `-a` 选项检测二进制文件：

   `discover -a` *binary_name*

   注 – 必须使用 Oracle Solaris Studio 12.3 中的 Discover 版本。`-a` 选项在早期的 Discover 版本中不可用。

   ---

2. 运行已检测过的二进制文件。动态内存访问数据将被写入 *binary_name*`.analyze` 目录的 `dynamic` 子目录中。

注 – 有关在使用 Discover 检测二进制文件时可以指定的其他检测选项，请参见《Oracle Solaris Studio 12.3： Discover 和 Uncover 用户指南》中的“检测选项”或 `discover` 手册页。可以将 `-c`、`-F`、`-N` 或 `-T` 选项与 `-a` 选项结合使用。

收集动态内存访问数据之后，可以启动代码分析器 GUI 来分析和显示这些数据以及先前可能已收集的任何静态代码数据（请参见第 16 页中的 “启动代码分析器 GUI”）。或者，可以使用未经检测的二进制文件副本来收集代码覆盖数据。

收集代码覆盖数据

收集 C 或 C++ 程序的代码覆盖数据的过程包含以下三个步骤：使用 Uncover 检测二进制文件，运行检测过的二进制文件，然后再次运行 Uncover 以生成覆盖报告供代码分析器使用。

可以在检测二进制文件后多次运行该检测过的二进制文件，并累积各次运行所获取的数据，然后生成覆盖报告。

要使用 Uncover 检测程序以收集数据供代码分析器使用，必须已使用 Oracle Solaris Studio 12.3 C 或 C++ 编译器对程序进行了编译。使用 `-g` 选项进行编译可生成调试信息，从而使代码分析器可以使用源代码级别覆盖信息。

注 – 如果在编译程序以便使用 Discover 进行检测时保存了二进制文件副本，可以将该副本重命名为原始二进制文件名，以供在使用 Uncover 进行检测时使用。例如：

```
cp a.out.save a.out
```

从二进制文件中收集代码覆盖数据：

1. 使用 Uncover 检测二进制文件：

   ```
   uncover binary_name
   ```

2. 运行检测过的二进制文件一次或多次。代码覆盖数据将被写入 *binary_name*`.uc` 目录。
3. 结合使用 Uncover 与 `-a` 选项基于累积的数据生成代码覆盖报告：

   ```
   uncover -a  binary_name.uc
   ```

   覆盖报告将被写入 *binary_name*`.analyze` 目录的 coverage 子目录中。

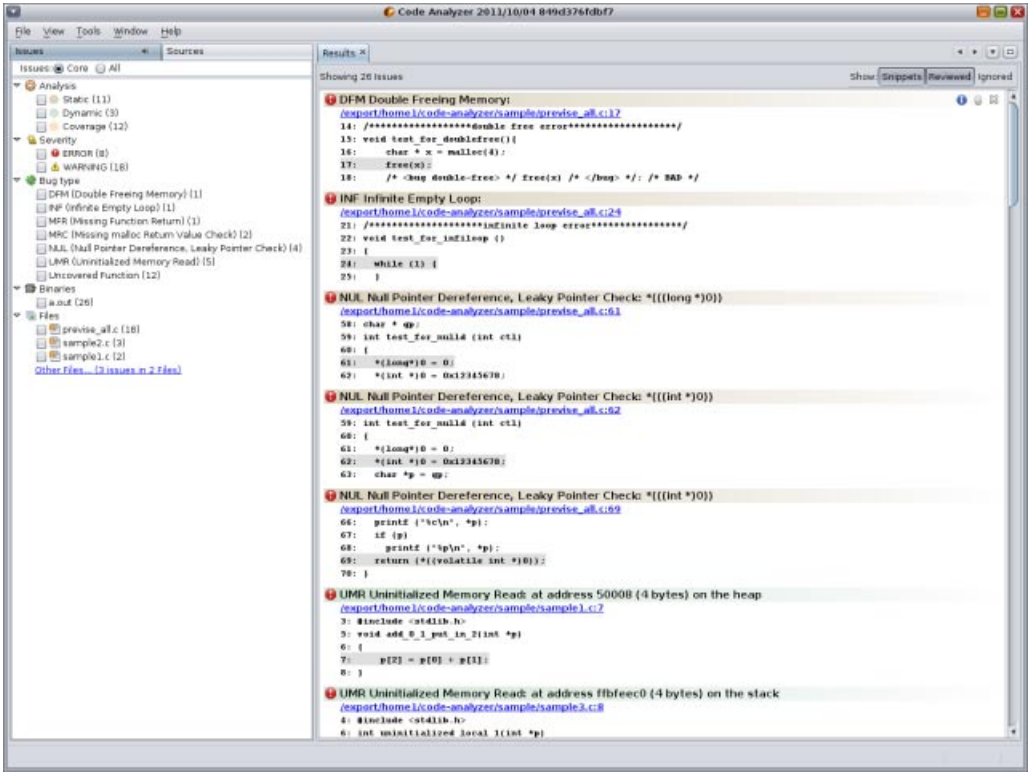
注 – 必须使用 Oracle Solaris Studio 12.3 中的 Uncover 版本。 `-a` 选项在早期的 Uncover 版本中不可用。

启动代码分析器 GUI

您可以使用代码分析器 GUI 分析其中一种、两种或全部三种类型的数据。要启动 GUI，请键入 `code-analyzer` 命令以及要为其分析所收集错误数据的二进制文件的路径：

`code-analyzer` *binary_name*

此时将打开代码分析器 GUI 并显示 *binary_name*`.analyze` 目录中的数据。



运行代码分析器 GUI 时，可以转换为显示针对其他二进制文件收集的数据，这可以通过选择 "Open"（打开）> "File"（文件）并导航到该二进制文件来实现。

GUI 中的联机帮助介绍了如何使用所有功能来过滤显示的结果、显示或隐藏问题以及显示有关特定问题的更多信息。《Oracle Solaris Studio 12.3 代码分析器教程》通过使用一个样例程序来指导您完成完整的数据收集和分析方案。

◆ ◆ ◆ 附录 A

代码分析器分析的错误

编译器、Discover 和 Uncover 可在您的代码中查找静态代码问题、动态内存访问问题以及覆盖问题。以下各节列出了可由这些工具找到并由代码分析器分析的特定错误类型。

静态代码问题

静态代码检查可查找以下类型的错误：

- ABR：数组越界读
- ABW：数组越界写
- DFM：双重释放内存
- ECV：显式强制类型转换违规
- FMR：读取释放的内存
- FMW：写入释放的内存
- INF：无限空循环
- 内存泄漏
- MF：返回缺少的函数
- MRC：缺少 malloc 返回值检查
- NFR：返回未初始化的函数
- NUL：NULL 指针解除引用，泄漏指针检查
- RFM：返回释放的内存
- UMR：未初始化的内存读取，未初始化的内存读取位操作
- URV：未使用的返回值
- VES：超出范围的局部变量使用

动态内存访问问题

动态内存访问检查可查找以下类型的错误：

- ABR：数组越界读
- ABW：数组越界写
- BFM：释放错误的内存块
- BRP：错误的重新分配地址参数
- CGB：损坏的保护块
- DFM：双重释放内存
- FMR：读取释放的内存
- FMW：写入释放的内存
- IMR：无效的内存读取
- IMW：无效的内存写入
- 内存泄漏
- OLP：重叠源和目标
- PIR：部分初始化的读取
- SBR：堆栈越界读
- SBW：堆栈越界写
- UAR：读取未分配的内存
- UAW：写入未分配的内存
- UMR：读取未初始化的内存

动态内存访问检查可查找以下类型的警告：

- AZS：分配零大小
- 内存泄漏
- SMR：推测性未初始化内存读取

代码覆盖问题

代码覆盖检查可确定哪些函数未被覆盖。在结果中，发现的代码覆盖问题会标记为 "Uncovered Function"（未覆盖的函数），并且带有潜在覆盖百分比，此百分比是指在添加覆盖相关函数的测试时将添加到应用程序总覆盖的覆盖百分比。

索引

要

优