Oracle® Developer Studio 12.5：代码分析器教程

2016 年 6 月

文件号码 E71976

本教程使用样例程序说明如何使用 Oracle Developer Studio 编译器、`discover` 内存错误搜索工具、`uncover` 代码覆盖工具以及代码分析器 GUI 来查找和更正常见的编程错误、动态内存访问错误以及代码覆盖问题。

代码分析器简介

Oracle Developer 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。它与 Oracle Developer Studio 编译器、内存错误搜索工具 `discover` 以及代码覆盖工具 `uncover` 配合使用。

代码分析器包含以下三种类型的分析：

- 编译过程中的静态代码检查
- 动态内存访问检查
- 代码覆盖分析

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。新编译器选项利用 Oracle Developer Studio 编译器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。

代码分析器利用 `discover` 收集的动态内存数据来发现应用程序运行时与内存相关的错误。它使用 `uncover` 收集的数据来测量代码覆盖。

除了提供对各个单独的分析类型的访问以外，代码分析器还将静态代码检查与动态内存访问分析和代码覆盖分析集成，可以发现应用程序中单独使用其他错误检测工具无法发现的许多重要错误。

获取样例应用程序

可从 Oracle Developer Studio 12.5 下载 Web 页面上的样例应用程序 zip 文件中获取该样例程序的源代码，网址为 `http://www.oracle.com/technetwork/server-storage/solarisstudio/downloads/index.html`。在接受了许可证并下载之后，可以将其在所选的目录中解压缩。

`sample` 应用程序位于 `OracleDeveloperStudio12.5-Samples` 目录的 `CodeAnalyzer` 子目录中。

`sample` 目录包含以下源代码文件：

```
main.c
previse_1.c
previse_all.c
sample1.c
sample2.c
sample3.c
sample4.c
```

收集和显示数据

您可以使用代码分析器工具最多收集三种类型的数据。

收集和显示静态错误数据

使用 -xprevise 编译器选项生成二进制文件时，编译器会自动提取静态错误并将数据放入源代码所在目录中的 *binary-name*.analyze 目录的 static 子目录中。有关编译器找到的静态错误的类型列表，请参见“静态代码问题” [15]。

1. 在 sample 目录中，键入以下内容以生成应用程序：

   注 - -xprevise 选项与 -xanalyze=code 同义，后者已弃用。

   - 在 Oracle Solaris 上：

     ```
     $ cc -xprevise main.c previse*.c sample1.c sample2.c sample3.c
     ```

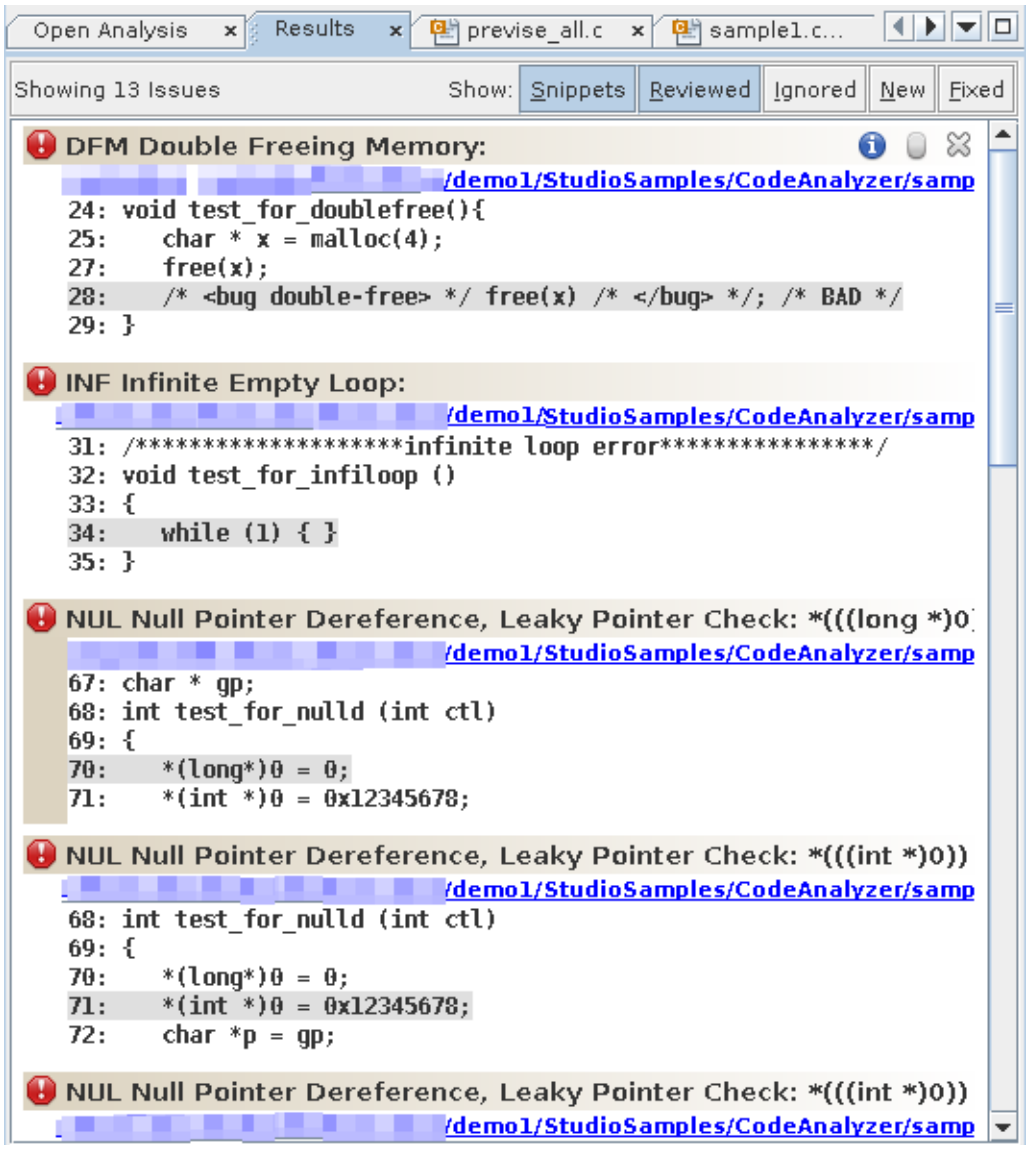
   - 在 Oracle Linux 上：

     ```
     $ cc -xannotate -xprevise main.c previse*.c sample1.c sample2.c sample3.c
     ```

   注 - 教程的此部分无需编译 sample4.c。

   静态错误数据将写入 sample/a.out.analyze/static 目录。
2. 启动代码分析器 GUI 以查看结果：

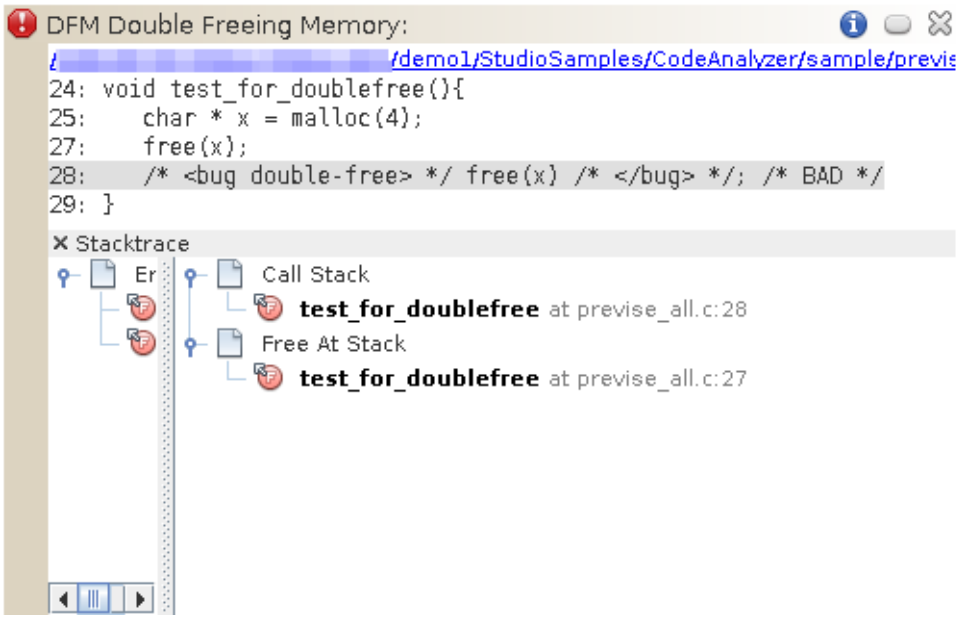
   ```
   $ code-analyzer a.out &
   ```
3. 此时将打开代码分析器 GUI，"Results"（结果）标签中会显示编译期间发现的静态代码问题。"Results"（结果）标签左上的文本指示发现了十三个静态代码问题。



对于每个问题，此标签显示问题类型、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。

4. 要查看有关第一个问题（"Double Freeing Memory"（双重释放内存）错误）的更多信息，请单击错误图标 。

   此时将打开问题的堆栈跟踪，其中显示错误路径：



请注意，在打开堆栈跟踪时，问题的右上角的图标从 变为 ，指示您已经查看了该问题。

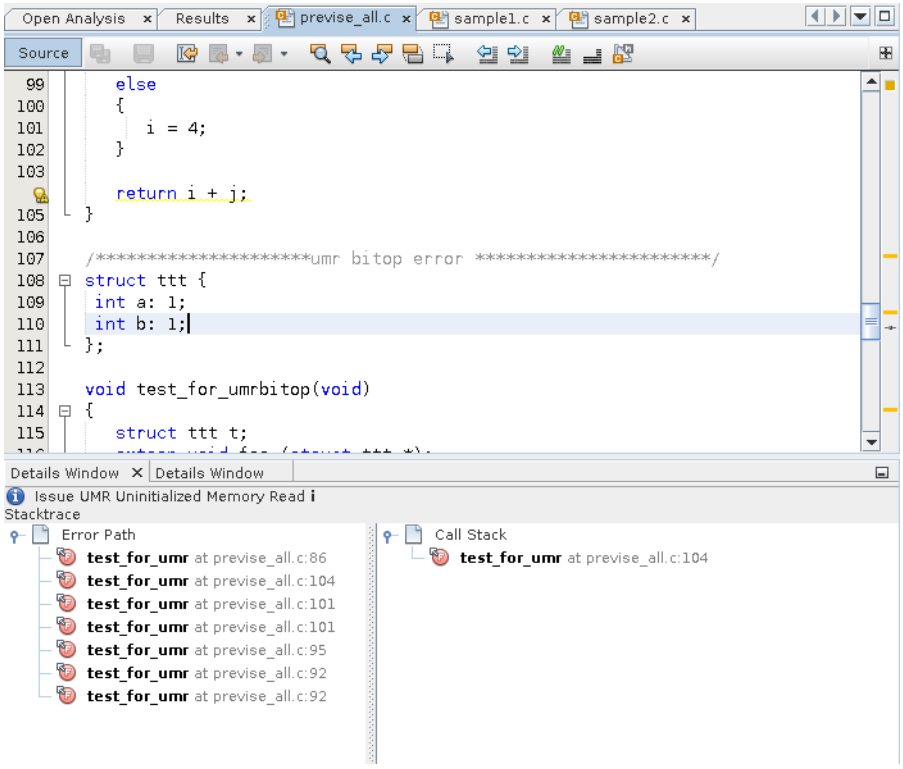
注 - 通过单击 "Results"（结果）标签顶部的 "Reviewed"（已查看）按钮 Reviewed，可隐藏您已经查看的问题。再次单击此按钮可取消隐藏问题。

---

5. 单击同一个错误图标以关闭堆栈跟踪。
6. 单击某个 "Uninitialized Memory Read"（读取未初始化的内存）警告的堆栈跟踪警告图标 可打开堆栈跟踪。

   此问题的错误路径所包含的函数调用远远多于 "Double Freeing Memory"（双重释放内存）问题的错误路径所包含的函数调用。
7. 双击第一个函数调用。

   此时将打开源文件，并且突出显示该调用。错误路径显示在源代码下的 "Details Window"（详细信息窗口）中。



8. 双击错误路径中的其他函数调用以沿着路径找到导致错误的代码。
9. 单击问题描述左边的信息按钮 可了解有关 UMR 错误类型的更多信息。

   联机帮助浏览器中显示了错误类型的描述，其中包括代码示例和可能的原因。
10. 按右上角的 X 关闭代码分析器 GUI。

收集和显示动态内存使用情况数据

无论是否收集了静态数据，您都可以编译、检测和运行应用程序来收集动态内存访问数据。有关通过使用 `discover` 检测应用程序然后运行应用程序找到的动态内存访问错误的列表，请参见“动态内存访问问题” [16]。

1. 在 `sample` 目录中，使用 `-g` 选项生成样例应用程序。

   此选项可生成调试信息，从而使代码分析器可以显示错误和警告的源代码和行号信息。

   - 在 Oracle Solaris 上：

     ```
     $ cc -g main.c previse*.c sample1.c sample2.c sample3.c
     ```

   - 在 Oracle Linux 上：

     ```
     $ cc -xannotate -g main.c previse*.c sample1.c sample2.c sample3.c
     ```

   注 - 教程的此部分无需编译 `sample4.c`。

   ---

2. 请保存该二进制文件的副本以供收集覆盖数据时使用，因为无法检测已经检测的二进制文件。

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

3. 使用 `discover` 检测二进制文件。

```
$ discover -a a.out
```

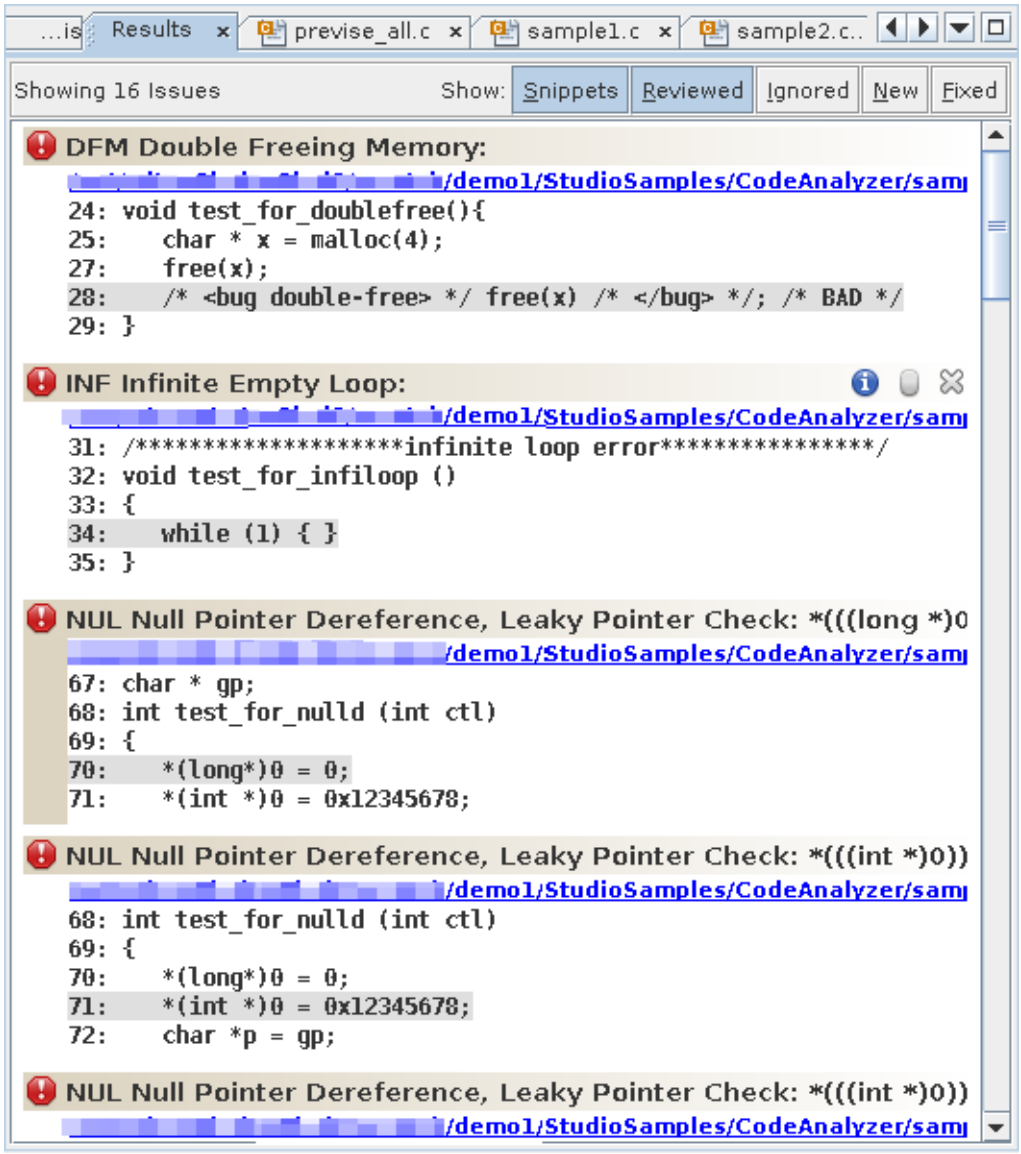
4. 运行检测过的二进制文件以收集动态内存访问数据。

```
$ ./a.out
```

动态内存访问错误数据将写入 `sample/a.out.analyze/dynamic` 目录。

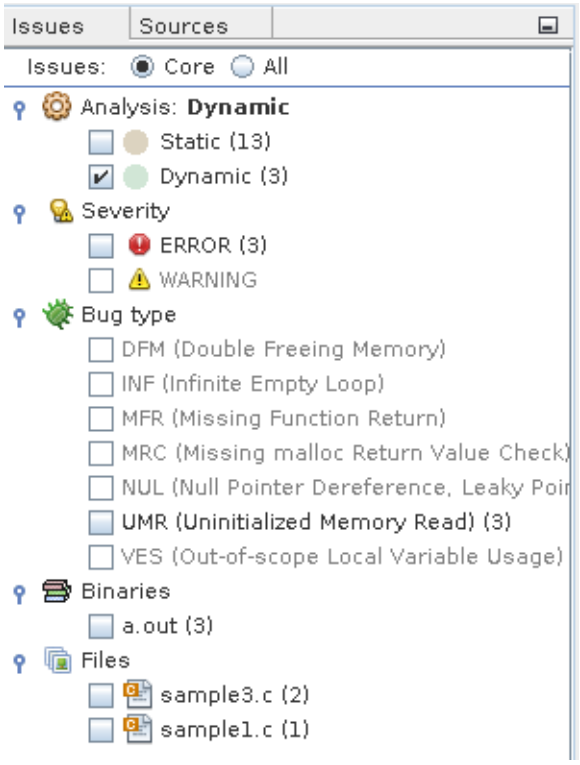
5. 启动代码分析器 GUI 以查看结果。

```
$ code-analyzer a.out &
```



"Results"（结果）标签同时显示静态问题和动态内存问题。问题描述的背景颜色可以指明问题是静态代码问题（棕褐色）还是动态内存访问问题（淡绿色）。

6. 要过滤结果以便仅显示动态内存问题，请在 "Issues"（问题）标签中选中 "Dynamic"（动态）选项。

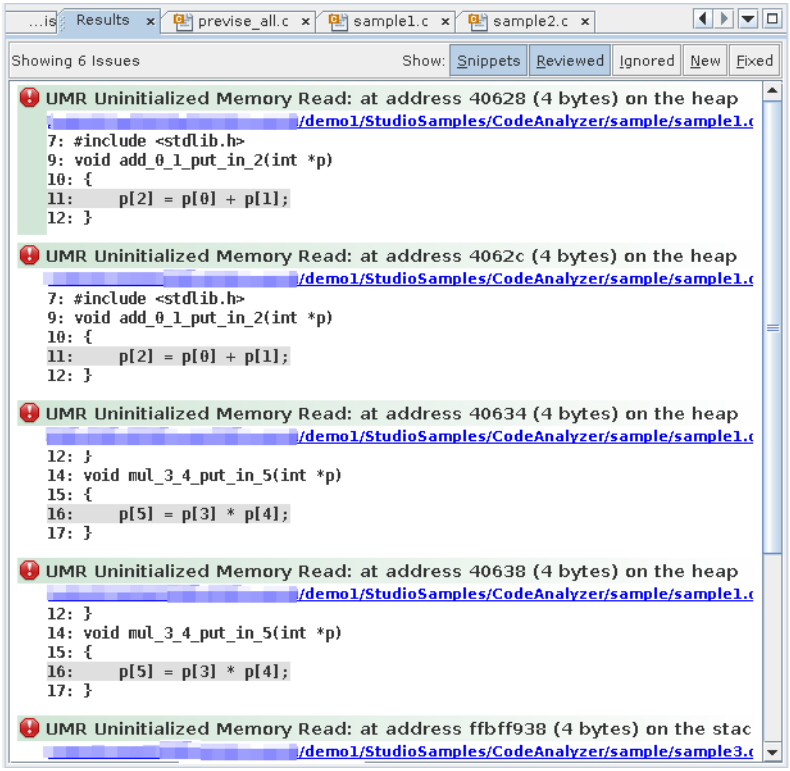


现在，"Results"（结果）标签仅显示三个核心动态内存问题。

注 - 修复核心问题后可能会消除其他问题。一个核心问题通常会伴有 "All"（所有）视图中列出的多个问题，例如，因为这些问题具有共同的分配点或出现在同一函数中的同一数据地址上。

---

7. 要查看所有动态内存问题，请选择 "Issues"（问题）标签顶部的 "All"（所有）单选按钮。现在，"Results"（结果）标签显示六个动态内存问题。



查看已添加到显示屏幕中的三个问题，并了解它们如何与核心问题相关联。在基于相应成因修复显示屏幕中的第一个问题后，可能还会消除第二个和第三个问题。

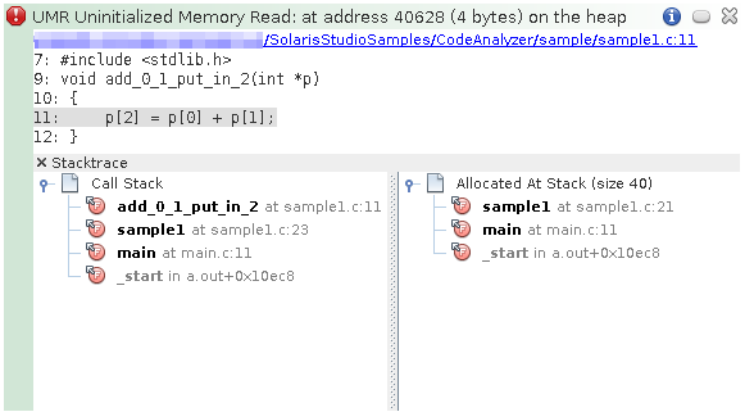
要在调查第一个问题的成因时隐藏其他动态内存访问问题，请单击每个问题对应的 "Ignore"（忽略）按钮 。

注 - 以后，通过单击 "Results"（结果）标签顶部的 "Ignored"（已忽略）按钮，可以重新显示关闭的问题。

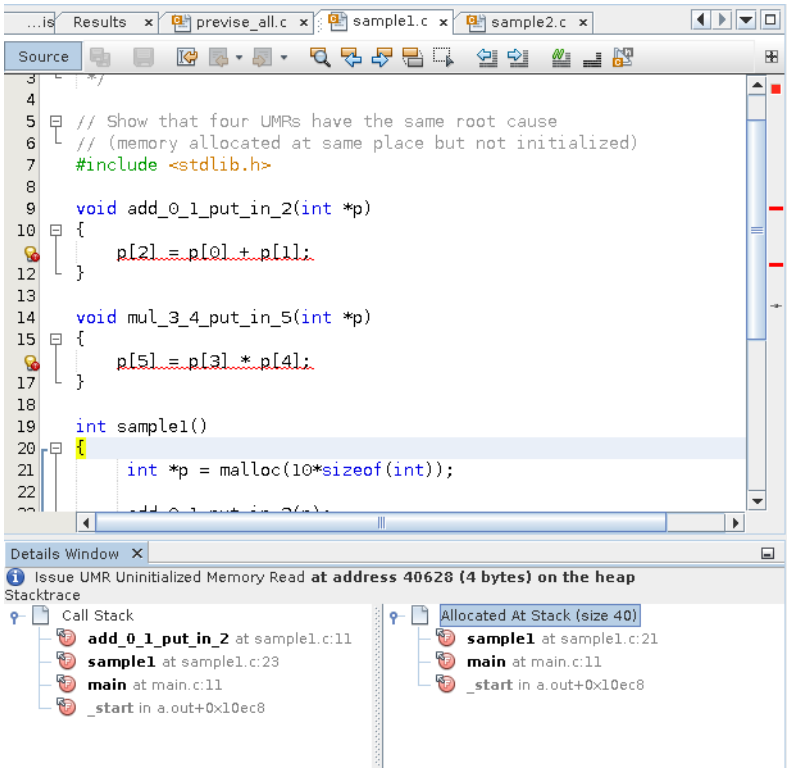
---

8. 通过单击错误图标以显示堆栈跟踪来调查第一个问题。

   对于此问题，堆栈跟踪包括 "Call Stack"（调用堆栈）和 "Allocated At Stack"（堆栈上的已分配空间）。



9. 双击堆栈中的函数调用以查看源文件中的关联行。
   打开源文件时，堆栈跟踪显示在该文件下的 "Details Window"（详细信息窗口）中。



10. 按右上角的 X 关闭代码分析器 GUI。

收集和显示代码覆盖数据

无论是否收集了静态数据或动态内存访问数据，您都可以编译、检测和运行应用程序来收集代码覆盖数据。请注意，在收集动态内存错误数据之前使用 -g 选项生成应用程序时，您在检测二进制文件之前保存了它的一个副本。

1. 复制要检测的已保存二进制文件以进行覆盖数据收集。

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

2. 使用 uncover 检测二进制文件：

```
$ uncover a.out
```

3. 运行检测过的二进制文件以收集代码覆盖数据。

```
$ ./a.out
```

代码覆盖数据将写入 `sample` 目录中的 `a.out.uc` 目录。

4. 对 `a.out.uc` 目录运行 `uncover`。

```
$ uncover -a a.out.uc
```

代码覆盖数据将写入 `sample/a.out.analyze/uncover` 目录。

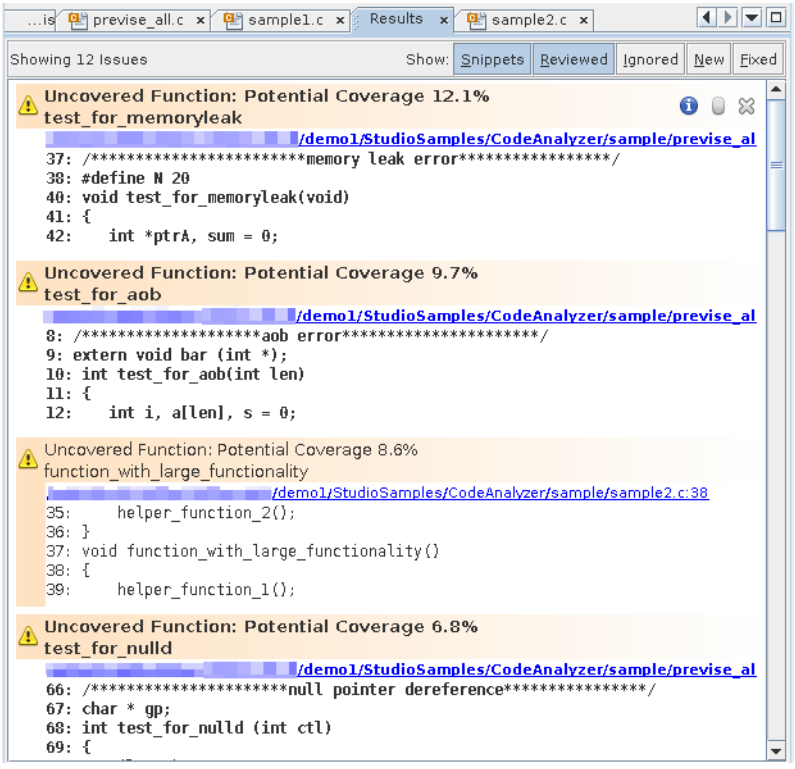
5. 启动代码分析器 GUI 以查看结果：

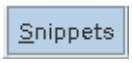
```
$ code-analyzer a.out &
```

"Results"（结果）标签同时显示静态问题、动态内存问题和代码覆盖问题。

6. 要过滤结果以便仅显示代码覆盖问题，请在 "Issues"（问题）标签中选中 "Coverage"（覆盖）选项。

现在，"Results"（结果）标签仅显示十二个代码覆盖问题。每个问题的描述中均包含一个潜在的覆盖百分比，此百分比是在增加覆盖相关函数的测试时将加入该应用程序总覆盖范围的覆盖百分比。

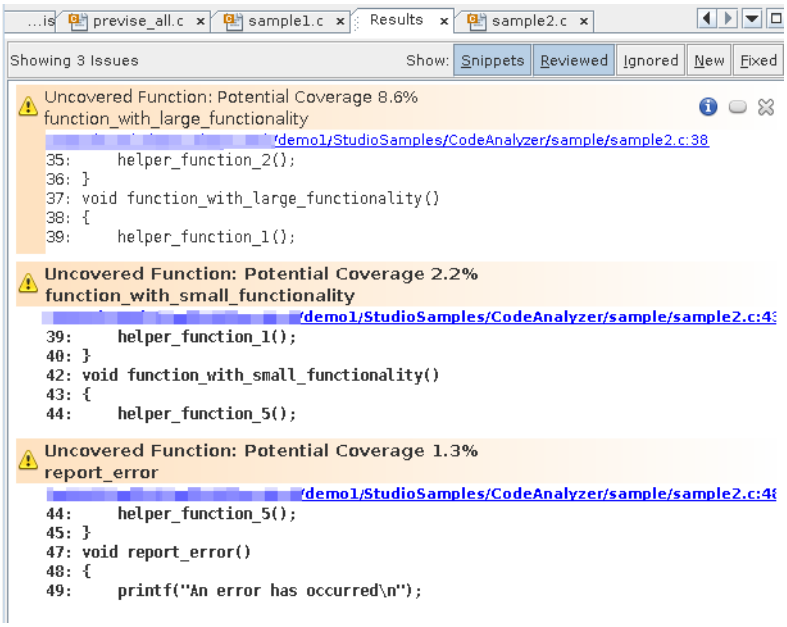


提示 - 要查看所有问题而不上下滚动，请单击 "Results"（结果）标签顶部的 "Snippets"（代码片段）按钮  以隐藏代码片段。

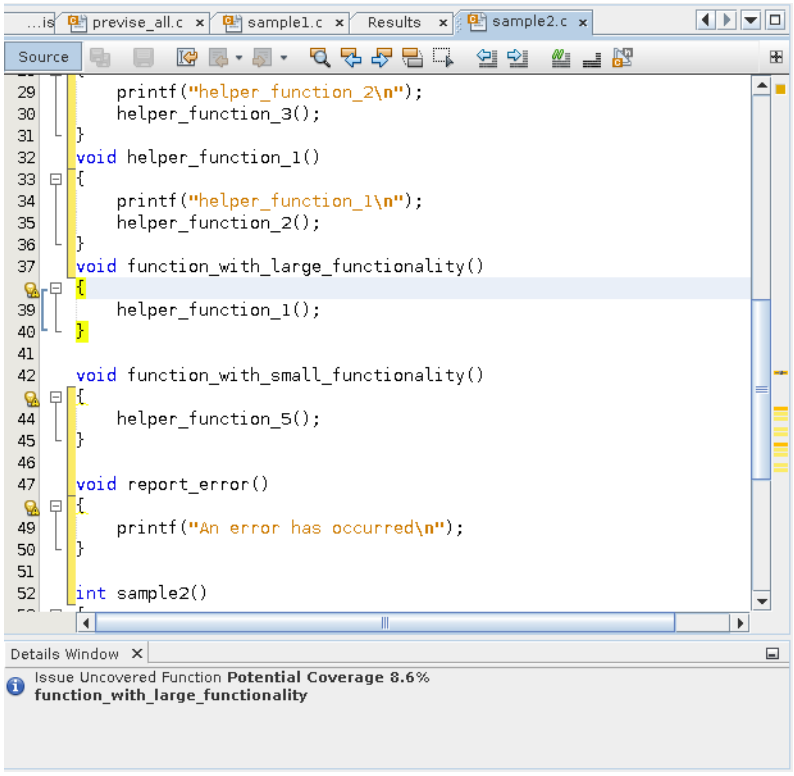
在 "Issues"（问题）标签中，`previse_all.c` 源文件中有九个覆盖问题，`sample2.c` 源文件中有三个覆盖问题，而 `previse_1.c` 源文件中有一个覆盖问题。

7. 要仅显示 `sample2.c` 文件的问题，请在 "Issues"（问题）标签上选中该文件对应的选项。

   "Results"（结果）标签现在仅显示在 `sample2.c` 中找到的三个代码覆盖问题。



8. 单击其中一个问题的源文件路径链接以打开源文件。在源文件中向下滚动，直到在左边界看到警告图标。



将使用黄色的方括号来标记未覆盖的代码。

在文件中找到的覆盖问题标记有警告图标 。

使用 codean 命令行工具

类似地，可以通过 codean 命令使用代码分析器中的所有功能。本部分是有关如何使用 codean 命令发现代码中的新静态代码问题的简短教程，采用了 OracleDeveloperStudio12.5-Samples 中的同一 sample 程序。

1. 本教程前面的各个部分未编译 sample4.c。可使用 cat 命令预览此文件。

```
$ cat sample_4.c
int another_new_umr()
{
  int i;
  if (i)
    return 0;
  else
    return 1;
}
```

请注意，int i 未初始化。

2. 编译源代码并生成静态报告。

在 Oracle Solaris 上：

```
$ cc -g -xprevise main.c previse_1.c previse_all.c sample1.c sample2.c sample3.c
```

在 Oracle Linux 上：

```
$ cc -xannotate -g -xprevise main.c previse_1.c previse_all.c sample1.c sample2.c sample3.c
```

3. 使用 `codean --save` 选项将静态报告保存到 `a.out`。

```
$ codean --save -s a.out
```

4. 重新编译 `sample` 应用程序（这次包括 `sample4.c`）。

在 Oracle Solaris 上：

```
$ cc -g -xprevise *.c
```

在 Oracle Linux 上：

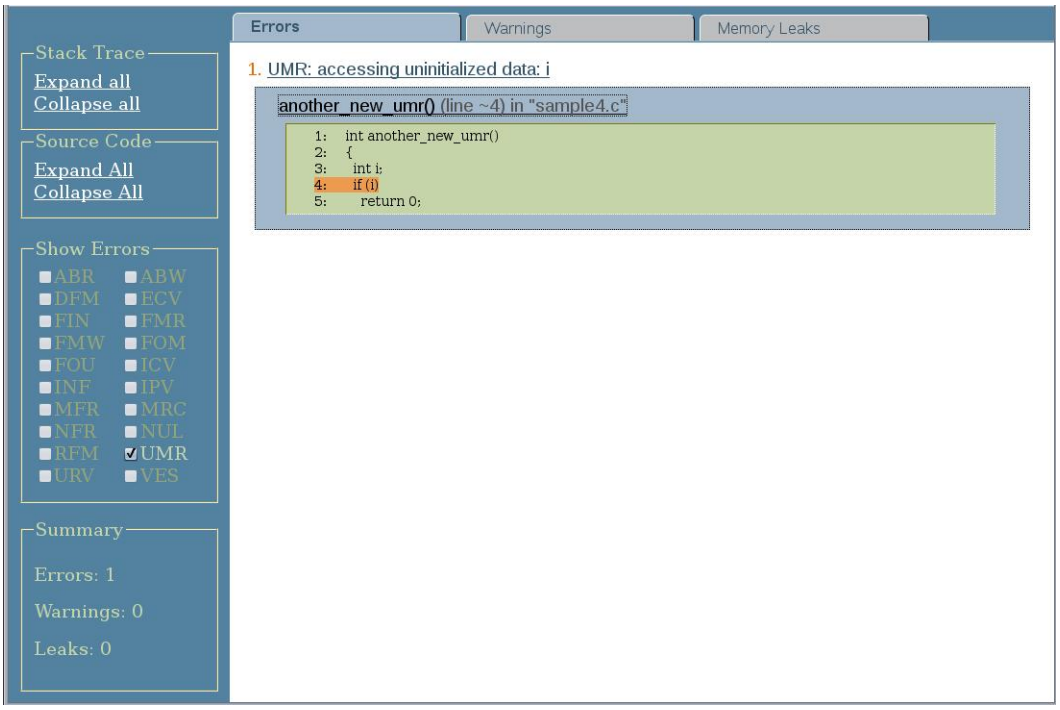
```
$ cc -g -xannotate -xprevise *.c
```

此新函数从来未从 `main` 调用过，但它将引入一个新的 UMR 错误。

5. 使用 `--whatisnew` 选项可获取有关新增加的静态问题的报告。

```
$ codean --whatisnew -s a.out
STATIC report of a.out showing new issues:
Compare the latest results against a.out.analyze/history/2014.8.4.14.49.56...
ERROR 1 (UMR): accessing uninitialized data: i at:
        another_new_umr()  <sample_4.c : 4>
                1:      int another_new_umr()
                2:      {
                3:        int i;
                4:=>      if (i)
                5:          return 0;
PREVISE SUMMARY for a.out: 1 new error(s), 0 new warning(s), 0 new leak(s) in total
```

下图展示了有关 `codean` 生成的静态代码问题的 HTML 报告。



有关 codean 的更多信息，请参见《Oracle Developer Studio 12.5：代码分析器用户指南》 中的 “使用代码分析器命令行工具 (codean)”和 codean(1) 手册页。

利用代码分析器找到的问题来改进代码

通过修复代码分析器找到的核心问题，应该能够消除在代码中找到的其他问题并大大提高代码的质量和稳定性。

虽然静态错误检查可以发现应用程序中有风险的代码，但也可能存在误报。动态检查可帮助检验并消除这些错误，使您能够更准确地了解代码中存在的问题。代码覆盖检查还可帮助改进您的动态测试套件。

代码分析器整合了这三种检查类型的结果，使您能够在一个工具中对所有代码进行最准确的分析。

代码分析器工具找到的潜在错误

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。本部分列出了由这些工具发现并由代码分析器分析的特定错误类型。

有关这些错误和警告的更多信息，请参见 《Oracle Developer Studio 12.5：代码分析器用户指南》 中的 附录 A, “代码分析器分析的错误,”。

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
- MLK：内存泄漏
- MFR：缺少函数返回值
- 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：读取未初始化的内存。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
- FRP：释放的重新分配参数
- IMR：无效的内存读取
- IMW：无效的内存写入
- MLK：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
- MLK：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后要添加到应用程序总覆盖中的覆盖百分比。

Oracle Developer Studio 12.5：代码分析器教程

文件号码 E71976

文档可访问性

有关 Oracle 对可访问性的承诺，请访问 Oracle Accessibility Program 网站 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc。

获得 Oracle 支持

购买了支持服务的 Oracle 客户可通过 My Oracle Support 获得电子支持。有关信息，请访问 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info；如果您听力受损，请访问 http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs。

Part No: E71976

ORACLE