Cppcheck 1.64 中文使用手册

Author: Daniel Marjamäki

Translator： XiaoH

Email: nightwishxiaoh@gmail.com

Github: https://github.com/danmar

目录

第一章 序

Cppcheck 是一个 C/C++代码分析工具。它不像 C/C++编译器以及许多其他类型的代码分析工具，Cppcheck 不检查语法错误，它仅监测通常编译器无法识别的代码 bug（也就是所谓的逻辑错误）。我们的目标是没有误报。

支持的代码及平台：

- 支持包括多种编译器扩展的非标准代码，内嵌汇编代码等。
- Cppcheck 与任何支持 C++最新标准的编译器兼容。
- Cppcheck 能够在任何有足够内存和 CPU 的平台上运行。

精确度

你要明白 Cppcheck 也是有极限的（Cppcheck 不是万能的，但是没 Cppcheck 是万万不能的…）。一般几乎不会出现误报的问题，但仍有许多检查不出来的代码 bug。

比起用 Cppcheck，只要你足够认真地测试你的软件，你将会发现更多 bug。

比起用 Cppcheck，只要你足够耐心地监控你的软件，你也会发现更多 bug。

但用 Cppcheck，你会发现很多完全靠你个人测试会漏掉的 bug，就是这样。

第二章 骑兵列队，准备突围

2.1 第一个测试例子

一段简单的代码：

```
int main()
{
    char a[10];
    a[10] = 0;
    return 0;
}
```

之后保存并执行：

```
cppcheck file1.c
```

这时 cppcheck 将会输出如下信息：

```
Checking file1.c…
[file1.c:4] : (error) Array ’a[10]’ index 10 out of bounds
```

数组下标越界。就是这样，你写代码的时候可能就忘了，靠你自己检查如果是上述的例子可能需要 1 秒钟检查，那如果是 2000 行，200 个文件呢？你还行不？

2.2 检查一个文件夹下的所有文件

一般而言一个程序会有很多源文件，你需要将它们都检查一遍。Cppcheck 能够用如下命令检查一个指定目录下的所有源文件：

```
cppcheck path
```

如果 path 是一个文件夹，cppcheck 将会检查指定文件夹下所有源文件。

Checking path/file1.cpp...

1/2 files checked 50% done

Checking path/file2.cpp...

2/2 files checked 100% done

2.3 在检查中排除文件或文件夹

有两种方法指定排除检查的文件或文件夹：

- 第一种是直接指定需要检查的路径和文件，如下所示：

  cppcheck　src/a　src/b

  此命令直接检查这两个目录下的所有文件。

- 第二种是使用-i 选项来指定忽略的文件或文件夹，如下所示，排除堆 src/c 目录下所有文件的检查：

  cppcheck　-isrc/c　src

2.4 严重性（Severities）

可能出现的严重性信息标签如下：

error

找到 bug 就打上这个标签。

warning

防御性编程建议，以避免 bug 的产生。

style

与代码整洁相关的格式问题（未使用的函数、冗余的代码、常量性等）

performance

代码优化提速建议。不过这也仅仅是根据常识和经验给出的，至于按照建议修改了之后到底能不能提速、提速是否明显，是不能保证的（通俗的说就叫不一定靠谱）。

portability

移植警告。代码在不同编译器下表现是不同的。

information

对于检察出的问题的信息。

2.5 允许消息

缺省状态下只显示 error 消息。使用—enable 选项能够开启更多的检查选项。

```
# enable warning messages
cppcheck --enable=warning file.c

# enable performance messages
cppcheck --enable=performance file.c

# enable information messages
cppcheck --enable=information file.c

# For historical reasons, --enable=style enables warning, performance,
# portability and style messages. These are all reported as "style" when
# using the old xml format.
cppcheck --enable=style file.c

# enable warning and information messages
cppcheck --enable=warning,information file.c
```

```
# enable unusedFunction checking. This is not enabled by --enable=style
# because it doesn't work well on libraries.
cppcheck --enable=unusedFunction file.c

# enable all messages
cppcheck --enable=all
```

请注意--enabale=unusedFunction 仅仅在浏览整个程序时使用，--enable=all 同样也是只针对浏览整个程序时，原因是用 unusedFunction 选项检察时，如果一个函数没有被调用那么就会产生 warning（因为可能在其他文件中调用）。

2.5.1 不确定检察

默认情况下，Cppcheck 只会输出确定的 error 消息，而使用--inconclusive 选项后，不确定的分析结果也将会被输出。

```
cppcheck --inconclusive path
```

显然这也有可能产生错误的 warning 信息，就像本身没有 bug 的程序也有可能被编译器爆出有 bug 一样。所以仅在允许误报 warning 的情况下使用这条命令。

2.6 将结果存入文件

很多情况下需要将检查结果另存到一个文件中。这时就和 shell 命令行一样，用个重定向符号就行了，将标准错误输出重定向到 err.txt 中：

```
cppcheck file1.c 2>err.txt
```

2.7 多线程检查

-j 选项指定你想用多少个线程。例如使用 4 个线程检查一个文件夹下文件：

cppcheck -j 4 path

第三章 预处理配置

默认情况下 cppcheck 将会检查所有的预处理配置（含有#error 宏的除外）。

可以使用-D 选项来改变这种默认设置。当使用-D 选项时，cppcheck 将只检查指定的配置。你可以使用--force 或--max-configs 来覆盖配置数目。

```
# check all configurations
cppcheck file.c

# only check the configuration A
cppcheck -DA file.c

# check all configurations when macro A is defined
cppcheck -DA --force file.c
```

另一个选项是-U，能够取消一个符号的定义，例如：

```
cppcheck -UX file.c
```

这意味着符号 X 没有定义，当定义了 X 时，cppcheck 将不会检查发生什么。

第四章 XML 输出

Cppcheck 能够以 XML 格式输出。请尽量使用新版的 XML 格式，老版的 XML 格式支持只是为了与以前的版本兼容。

新版的 XML 格式修正了老板的一些 bug，新版本格式以后也会一直随着 Cppcheck 的版本升级，使--xml 来开启老版 XML 格式支持。下面是如何使用 XML 格式输出检查信息的命令：

cppcheck --xml-version=2 file1.cpp

示例输出：

```
<?xml version="1.0" encoding="UTF-8"?>
<results version="2">
    <cppcheck version="1.53">
    <errors>
        <error id="someError" severity="error" msg="short error text"
            verbose="long error text" inconclusive="true">
            <location file="file.c" line="1"/>
        </error>
    </errors>
</results>
```

4.1 <error>标签

每个 error 都以<error>的形式输出，其属性如下：

属性	说明
id	error 的 id
severity	error, warning, style, performance, portability, information 其中之一
msg	短格式 error 信息
verbose	长格式 error 信息
inconclusive	仅在消息为不确定时使用

4.2 <location>标签

所有与 error 相关的地方都会以<location>的标签列出。

属性	说明
file	既可以是相对路径，也能是绝对路径
line	行号
msg	暂无，但以后的版本会对 location 有说明

第五章 重新格式化输出

如果你想重新格式化输出来显得优异一点，那就用模板吧。

例如，你喜欢 Visual Studio 的输出风格，请使用--template=vs 选项：

cppcheck --template=vs gui/test.cpp

输出就会形如：

Checking gui/test.cpp...

gui/test.cpp(31): error: Memory leak: b

gui/test.cpp(16): error: Mismatching allocation and deallocation: k

又比如使用 gcc 兼容的输出风格，--template=gcc

cppcheck --template=gcc gui/test.cpp

则输出如下：

Checking gui/test.cpp...

gui/test.cpp:31: error: Memory leak: b

gui/test.cpp:16: error: Mismatching allocation and deallocation: k

所以很明显，你可以定制自己的模板（例如一个自定义的逗号分隔的格式），举个栗子：

cppcheck --template="{file},{line},{severity},{id},{message}" gui/test.cpp

输出：

Checking gui/test.cpp...

gui/test.cpp,31,error,memleak,Memory leak: b

gui/test.cpp,16,error,mismatchAllocDealloc,Mismatching allocation and deallocation: k

支持以下格式说明符：

说明符	说明
callstack	如果可以的话，显示调用堆栈
file	文件名
id	消息 id
line	行号
message	冗长的消息文本
severity	严重程度

也支持转义字符：\b，\n，\r，\t

第六章 禁止（或者说屏蔽）

如果你想过滤掉一些 error，那么就丢了吧。

6.1 禁止特定的 error 类型

你可以禁止一些特定类型的 error。禁止 error 的格式为：

[error id]:[filename]:[line]

[error id]:[filename2]

[error id]

error id 就是你想要屏蔽的 error 类型的 id 号。获取 error id 最简单的方法就是使--xml 选项。从 XML 格式的输出复制粘贴，还可以用*号来禁止所有 warning。

文件名可以包含万金油匹配符“？”或“*”，*号匹配任何长度序列，？匹配任何单字符。强烈建议使用“/”作为任何系统的路径分隔符（而不是和 Windows 那么矫情的用“\”）。

6.1.1 命令行禁止

用--suppress=command line 选项来指明禁止的内容。示例：

cppcheck --suppress=memleak:src/file1.cpp src/

6.1.2 将禁止内容在文件中列出

新建一个关于禁止内容的文件，其中可以包含注释和空行：

// suppress memleak and exceptNew errors in the file src/file1.cpp

memleak:src/file1.cpp

exceptNew:src/file1.cpp

// suppress all uninitvar errors in all files

uninitvar

使用示例：

cppcheck --suppressions suppressions.txt src/

6.2 内联禁止

同样还可以直接在源代码中加入含有特定关键词的注释。注意，在加这些注释的时候，要考虑下对于代码可读性不要牺牲太大。

看下面一段代码将会产生一个 error 消息：

```
void f( ) {
    char arr[5];
    arr[10] = 0;
}
```

输出：

```
# cppcheck test.c
Checking test.c...
[test.c:3]: (error) Array 'arr[5]' index 10 out of bounds
```

为屏蔽这类 error，我们在源代码中直接加入一行注释，示例：

```
void f() {
    char arr[5];
    // cppcheck-suppress arrayIndexOutOfBounds
```

```
    arr[10] = 0;
}
```

现在如果使用--inline-suppr 选项，就不会有 error 信息了。

```
cppcheck --inline-suppr test.c
```

第七章 规则

代码审查器就是字符串玩具，要玩好这种玩具就是要对字符串的操作和原理神马的了然于心，然后不用我说也就应该知道了，可以用正则表达式自定义规则。

这些规则可能不能对代码进行极其精细的分析，但至少给了你一种匹配简单模式的检测方法。

想要了解如何编写规则：http://sourceforge.net/projects/cppcheck/files/Articles/

又一个示例：

```
<?xml version="1.0"?>
<rule>
    <tokenlist>LIST</tokenlist>
        <pattern>PATTERN</pattern>
            <message>
                <id>ID</id>
                    <severity>SEVERITY</severity>
                    <summary>SUMMARY</summary>
            </message>
</rule>
```

7.1 <tokenlist>

<tokenlist>是可选的，用这个标签可以控制检查的符号。

define

用于检查#define 的预处理描述。

raw

用于检查预处理输出。

normal

用于检查一般的符号表，有一些简化方式。

simple

用于检查简单符号表。

如果没有<tokenlist>标签，则默认开启 simple。

7.2 <pattern>

PATTERN 是会被执行的 PCRE（perl compatible regular expression）兼容的正则表达式

7.3 <id>

指明用户定义的消息 id。

7.4 <severity>

必须指定 information, performance, portability, style, warning, or error 之一.

7.5 <summary>

可选的，消息总结。

第八章 库的配置

当使用外部库时，例如 windows/posix/gtk/qt/等，Cppcheck 不知道外部函数的行为。cppcheck 将不能检测出内存泄露和缓存溢出，空指针引用等。但这可以通过配置文件解决。

如果你为一个流行的库写了一个配置文件，我想说：请联系我!!!（找我，找 cppcheck 的作者都行）

8.1 内存/资源泄露

示例：

```
void test()
{
    HPEN pen = CreatePen(PS_SOLID, 1, RGB(255,0,0));
}
```

代码中有明显的资源泄露，上述函数是 windows 中一个 API，使用完后需要释放资源。但 cppcheck 不假设从函数中返回的资源一定要被释放掉。所以将不会有 error 信息：

```
# cppcheck pen1.c
Checking pen1.c...
```

当然，如果你提供了 windows 配置文件，cppcheck 就能够发现这个 bug：

```
# cppcheck --library=windows.cfg pen1.c
Checking pen1.c...
[pen1.c:3]: (error) Resource leak: pen
```

示例 windows 配置文件：

```
<?xml version="1.0"?>
<def>
        <resource>
            <alloc>CreatePen</alloc>
            <dealloc>DeleteObject</dealloc>
        </resource>
</def>
```

8.2 函数参数：未初始化的内存

示例程序：

```
void test()
{
    char buffer1[1024];
    char buffer2[1024];
    CopyMemory(buffer1, buffer2, 1024);
}
```

这里的 bug 是 buffer2 没有初始化，因为你是要将 buffer2 中的东西拷到 buffer1 中，然而 Cppcheck 假设可以传递未初始化的参数。所以又是：

```
# cppcheck uninit.c
Checking uninit.c...
```

如果你提供了一个 windows 配置文件，那么就可以输出 bug 消息了：

```
# cppcheck --library=windows.cfg uninit.c
Checking uninit.c...
[uninit.c:5]: (error) Uninitialized variable: buffer2
```

一个针对 windows 的微型配置文件 windows.cfg:

```
<?xml version="1.0"?>
<def>
    <function name="CopyMemory">
        <arg nr="2">
            <not-uninit/>
        </arg>
    </function>
</def>
```

8.3 函数参数：空指针

Cppcheck 假设允许向函数传递空指针，示例程序：

```
void test()
{
    CopyMemory(NULL, NULL, 1024);
}
```

MSDN 的文档中并未对这种情形有明确的说明。但我们假设这种情况是不正确的，所以如果提供了 windows 配置文件，Cppcheck 就能够监测出此类 bug：

```
cppcheck --library=windows.cfg null.c
Checking null.c...
[null.c:3]: (error) Null pointer dereference
```

示例配置文件 windows.cfg：

```
<?xml version="1.0"?>
<def>
    <function name="CopyMemory">
        <arg nr="1">
            <not-null/>
        </arg>
    </function>
</def>
```

8.4 函数参数：格式化字符串

你可以定义一个需要格式化字符串的函数，可恶的栗子又来了：

```
void test()
{
    do_something("%i %i\n", 1024);
}
```

默认情况下不会有 error 信息：

```
# cppcheck formatstring.c
Checking formatstring.c...
```

创建类似如下的配置文件，告诉 Cppcheck 该字符串是一个格式化字符串：

```
<?xml version="1.0"?>
<def>
    <function name="do_something">
        <arg nr="1">
            <formatstr />
        </arg>
    </function>
</def>
```

现在 Cppcheck 就会报出 error 了：

```
cppcheck --library=test.cfg formatstring.c
Checking formatstring.c...
[formatstring.c:3]: (error) do_something format string requires 2 parameters but only 1 is g
```

8.5 函数参数：变量值的范围

可以定义值的合法性：

```
void test()
{
    do_something(1024);
}
```

首先这样是不会报出 error 信息的：

```
# cppcheck valuerange.c
Checking valuerange.c...
```

但在配置文件中可以描述 1024 越界，示例：

```
<?xml version="1.0"?>
<def>
    <function name="do_something">
        <arg nr="1">
            <valid>0-1023</valid>
        </arg>
    </function>
</def>
```

现在输出如下：

```
cppcheck --library=test.cfg range.c
Checking range.c...
[range.c:3]: (error) Invalid do_something() argument nr 1. The value is 1024 but the valid
```

8.6 noreturn（函数没有返回）

Cppcheck 不假设函数总会返回，示例：

```
void test(int x)
{
    int data, buffer[1024];
    if (x == 1)
        data = 123;
    else
        ZeroMemory(buffer, sizeof(buffer));
    buffer[0] = data; // <- error: data is uninitialized if x is not 1
}
```

理论上，如果 ZeroMemory 终止了程序就不会产生 bug，cppcheck 也不会报 error：

cppcheck noreturn.c

Checking noreturn.c...

然而，如果用—ckeck-library 和—enable=information 将得到以下输出：

cppcheck --check-library --enable=information noreturn.c

Checking noreturn.c...

[noreturn.c:7]: (information) --check-library: Function ZeroMemory( ) should have <noreturn>

若是提供了 windows.cfg 文件，bug 就会被检查出来：

cppcheck --library=windows.cfg noreturn.c

Checking noreturn.c...

[noreturn.c:8]: (error) Uninitialized variable: data

windows.cfg 文件示例：

```
<?xml version="1.0"?>
<def>
    <function name="ZeroMemory">
        <noreturn>false</noreturn>
    </function>
</def>
```

8.7 示例配置（strcpy()）

标准函数 strcpy()的配置文件：

```
<function name="strcpy">
<leak-ignore/>
    <noreturn>false</noreturn>
    <arg nr="1">
        <not-null/>
    </arg>
    <arg nr="2">
        <not-null/>
        <not-uninit/>
    </arg>
</function>
```

<leak-ignore/>告诉 Cppcheck 在泄露检测中忽略这个函数。将分配好的内存传递给这个函数并不意味着没有释放内存。

<noreturn>告诉 Cppcheck 此函数是否返回。

这个函数的第一个参数是一个指针，并且不能为空（null），所以就用<not-null>。

第二个参数同样也是一个不能为空的指针，而且指向的内存必须被初始化，使用<not-null>和<not-uninit>。

第九章 HTML 报告

可以将 XML 格式的输出转换为 HTML 的报告，前提是已经有了 Python 和 pygments 模块。Cppcheck 源码树中有专门的一个文件夹 htmlreport，下面包含了如何将 Cppcheck 的 XML 输出转换为 HTML 的脚本。

获取帮助信息：

htmlreport/cppcheck-htmlreport –h

输出：

Usage: cppcheck-htmlreport [options]

Options:

-h, --help show this help message and exit

--file=FILE The cppcheck xml output file to read defects from.
Default is reading from stdin.

--report-dir=REPORT_DIR
The directory where the html report content is written.

--source-dir=SOURCE_DIR
Base directory where source code files can be found.

用法示例：

./cppcheck gui/test.cpp --xml 2> err.xml

htmlreport/cppcheck-htmlreport --file=err.xml --report-dir=test1 --source-dir=.

第十章 图形用户接口

10.1 简介

Cppcheck 提供图形界面

10.2 开始检查源代码

点击 Check 菜单

10.3 查看结果

代码检查结果显示在 List 控件中，你可以通过 View 菜单来显示或者隐藏信息。并且结果可以保存在 XML 格式的文件中。

10.4 设置

Language 菜单提供了多语言支持，更多的功能请查看 Edit 中的 Preference(首选项)。

10.5 工程文件

工程文件用于针对于特定工程的设置。包括：

- 包含的文件
- 预处理定义