

将变量名作为字符串传递给具有默认参数的函数

问 9 年零 6 个月前 活跃 9 年零 6 个月前 浏览了 5000 次

假设有一个调试功能，这里简化为：

4

```
void DumpString(char* var, char* varname) {  
    printf("%s : '%s'\n", varname, var);  
}
```

★

1



```
char str[10]="foobar";  
DumpString(str, "str");
```

```
> str : foobar
```

让我们通过消除传递变量两次的不必要的额外要求来简化它，一次在引号中：

```
#define VARASSTR(v) #v  
  
void DumpString(char* var) {  
    printf("%s : '%s'\n", VARASSTR(var), var);  
}  
  
char str[10]="foobar";  
DumpString(str);  
  
> var : foobar
```

哎呀！它使用局部变量名称而不是传入的名称。让我们尝试一种不同的（不太理想的）方法：

```
#define DumpStr(v) DumpString(v, #v)  
  
void DumpString(char* var, char* varname) {  
    printf("%s : '%s'\n", varname, var);  
}  
  
char str[10]="foobar";  
DumpStr(str);  
  
> str : foobar
```

很好用。但是如果函数稍微复杂一点怎么办：

```
void DumpString(char* var, char* varname, int optionalvar=0) {  
    printf("%s : '%s'\n", varname, var);  
    printf("blah: %d", optionalvar);  
}
```

宏无法重载，因此 `DumpStr` 不起作用，我们已经排除了带有 `VARASSTR`。

如何处理（不诉诸多个类似但名称不同的函数/宏）？



“如何处理（不诉诸多个类似但名称不同的函数/宏）？”它不能。你不能重载宏。——微薄 ◆ 2012 年 4 月 2 日 0:11

您应该尝试一些具有反射功能的语言。不完全是。——耶利希 2012 年 4 月 2 日 0:13

1 您可以使用 [Variadic 宏技巧](#) 来“模拟”宏重载。——杰西·古德 2012 年 4 月 2 日 0:27 ✎

感谢您链接我的问题。:-) —— R.. [GitHub 停止帮助 ICE](#) 2012 年 4 月 2 日 1:15

你也不能重载函数（或者你为什么标记这个问题 [c](#)），那么问题是什么？——阿萨尔 2012 年 4 月 2 日 1:18

1 个回答

积极的

最老的

投票

这是非标准的，但在 GNU C 中作为扩展工作：

1

```
#define DumpStr(v, ...) DumpString(v, #v, ##__VA_ARGS__)
```

在 GNU C 中，您不能向可变参数宏传递任何参数，并且“标记粘贴运算符”`##` 在逗号和空可变参数列表之间应用时不会产生任何结果（因此尾随逗号被抑制）。

在 Visual C++ 中，我相信标记粘贴操作符 `##` 是不必要的（并且可能会破坏宏），因为如果它出现在空的参数列表之前，Visual C++ 会自动抑制尾随逗号。

请注意，使这种非标准的唯一原因是有时希望传递空参数列表。可变参数宏在 C99 和 C++11 中均已标准化。

编辑：这是一个不使用非标准功能的示例。你可以看到为什么有些人真的，真的希望在标准中解决这类问题：

```
#define DUMPSTR_1(v) DumpString(v, #v)
#define DUMPSTR_2(v, opt) DumpString(v, #v, opt)
#define DUMPSTR_NARG(...) DUMPSTR_ARG_N(__VA_ARGS__, 4, 3, 2, 1, 0)
#define DUMPSTR_ARG_N(_1, _2, _3, _4, n, ...) n
#define DUMPSTR_NC(f, ...) f(__VA_ARGS__)
#define DUMPSTR_NB(nargs, ...) DUMPSTR_NC(DUMPSTR_ ## nargs, __VA_ARGS__)
#define DUMPSTR_NA(nargs, ...) DUMPSTR_NB(nargs, __VA_ARGS__)
#define DumpStr(...) DUMPSTR_NA(DUMPSTR_NARG(__VA_ARGS__), __VA_ARGS__)
```

可能有一些更清洁的方法可以做到这一点。但没有那么多。

编辑 2：这是另一个不使用非标准功能的示例，由 [R 提供](#)。

```
#define STRINGIFY_IMPL(s) #s
#define STRINGIFY(s) STRINGIFY_IMPL(s)
```

```
#define ARG1_IMPL(a, ...) a
#define ARG1(...) ARG1_IMPL(__VA_ARGS__, 0)
#define DumpStr(...) DumpString(Stringify(ARG1(__VA_ARGS__)), __VA_ARGS__)
```

请注意，这需要 `DumpString` 更改的参数顺序，以便字符串化的函数名称是第一个参数。

分享 跟随

17年5月23日 12:24编辑

2012年4月2日 0:18回答



社区 机器人

1 • 1



约翰·卡尔斯贝克

34.6k ● 7 ● 90 ● 100

这个问题有解决方案，没有像这样的非标准黑客。 —— R.. [GitHub 停止帮助 ICE](#) 2012年4月2日 0:30

- 1 特别是，您将整个参数列表包含到中 `...`，然后使用宏从中提取第一个（或您想要的任何索引）参数 `__VA_ARGS__`。 —— R.. [GitHub 停止帮助 ICE](#) 2012年4月2日 0:32

@R.. 是的，有。并且哇是可怕的。我添加了一个我鼓吹的独立方式。你能推荐一个更好的吗？
—— [约翰·卡尔斯贝克](#) 2012年4月2日 0:54

我没有使用过它，但我相信如果你愿意，`boost` 预处理器会为你隐藏一些丑陋的东西。 —— [杰西·古德](#)
2012年4月2日 1:01

@Jesse 是的，它会以大量笨拙的依赖为代价。在这与不符合标准之间，我将采用单线解决方案。（假设它适用于您的编译器。） —— [约翰·卡尔斯贝克](#) 2012年4月2日 1:03

@R.. 我想不出任何更好的方法来做到这一点，而不会遇到关于参数太多或太少的警告.....?
—— [约翰·卡尔斯贝克](#) 2012年4月2日 1:09

我在想类似的事情 `#define DumpStr(...) DumpString(Stringify(ARG1(__VA_ARGS__)), __VA_ARGS__)`（请注意，`DumpString` 那时必须更改参数的顺序，但无论如何都不应该直接使用该函数）。 —— R.. [GitHub 停止帮助 ICE](#) 2012年4月2日 1:15

@R.. 但是定义不会 `ARG1` 遇到同样的问题吗？它可能需要一个空的`__VA_ARGS__`列表，这是非标准的，或者使用像我这样的混乱实现。 —— [约翰·卡尔斯贝克](#) 2012年4月2日 1:16

`#define ARG1_X(a, ...) a` 并且 `#define ARG1(...) ARG1_X(__VA_ARGS__, 0)` 应该工作。
—— R.. [GitHub 停止帮助 ICE](#) 2012年4月2日 1:18

@R.. 啊，是的，这还不错。这只需要四个支持宏，而不是七个。（不知何故，在 C-land 中，我们认为这很优雅。） —— [约翰·卡尔斯贝克](#) 2012年4月2日 1:25

好吧，它们中的大多数都是可重用的，如果您经常使用预处理器，它们往往已经很方便了.....
—— R.. [GitHub 停止帮助 ICE](#) 2012年4月2日 3:26