学习用 doxygen 生成源码文档

Arpan Sen, 资深工程师, Studio B Productions

简介： 维护用 C/C++ 开发的遗留系统并添加新特性是一项艰难的任务。幸运的是，doxygen 可以帮助您完成这个任务。doxygen 是一种用于 C/C++、Java™、Python 和其他编程语言的文档系统。本文在 C/C++ 项目的上下文中讨论 doxygen 的特性，以及如何用 doxygen 定义的标记生成代码文档。

发布日期： 2008 年 10 月 13 日
级别： 中级
其他语言版本： 英文

维护用 C/C++ 开发的遗留系统并添加新特性是一项艰难的任务。这涉及几方面的问题：理解现有的类层次结构和全局变量，不同的用户定义类型，以及函数调用图分析等等。本文在 C/C++ 项目的上下文中通过示例讨论 doxygen 的几个特性。但是，doxygen 非常灵活，也可用于用 Python、Java、PHP 和其他语言开发的软件项目。本文的主要目的是帮助您从 C/C++ 源代码提取出信息，但也简要描述了如何用 doxygen 定义的标记生成代码文档。

安装 doxygen

有两种获得 doxygen 的方法。可以下载预编译的可执行文件，也可以从 SVN 存储库下载源代码并自己编译。清单 1 演示的是后一种方法。

清单 1. 安装和构建 doxygen 源代码

```
bash-2.05$ svn co https://doxygen.svn.sourceforge.net/svnroot/doxygen/trunk doxygen-svn

bash-2.05$ cd doxygen-svn
bash-2.05$ ./configure -prefix=/home/user1/bin
bash-2.05$ make

bash-2.05$ make install
```

注意，配置脚本把编译的源代码存储在 /home/user1/bin 中（进行编译后，会在 PATH 变量中添加这个目录），因为并非每个 UNIX® 用户都有写 /usr 文件夹的权限。另外，需要用 svn 实用程序下载源代码。

使用 doxygen 生成文档

使用 doxygen 生成源代码的文档需要执行三个步骤。

生成配置文件

在 shell 提示上，输入命令 **doxygen -g** 。这个命令在当前目录中生成一个可编辑的配置文件 *Doxyfile*。可以改变这个文件名，在这种情况下，应该调用 doxygen -g <*user-specified file name*>，见 清单 2。

清单 2. 生成默认的配置文件

```
bash-2.05b$ doxygen -g
Configuration file 'Doxyfile' created.
Now edit the configuration file and enter
  doxygen Doxyfile
to generate the documentation for your project
bash-2.05b$ ls Doxyfile
Doxyfile
```

编辑配置文件

配置文件采用 <TAGNAME> = <VALUE> 这样的结构，与 Make 文件格式相似。下面是最重要的标记：

- **<OUTPUT_DIRECTORY>：**必须在这里提供一个目录名，例如 /home/user1/documentation，这个目录是放置生成的文档文件的位置。如果提供一个不存在的目录名，doxygen 会以这个名称创建具有适当用户权限的目录。
- **<INPUT>：**这个标记创建一个以空格分隔的所有目录的列表，这个列表包含需要生成文档的 C/C++ 源代码文件和头文件。例如，请考虑以下代码片

段：

```
INPUT = /home/user1/project/kernel /home/user1/project/memory
```

在这里，doxygen 会从这两个目录读取 C/C++ 源代码。如果项目只有一个源代码根目录，其中有多个子目录，那么只需指定根目录并把 <RECURSIVE> 标记设置为 **Yes**。

- **<FILE_PATTERNS>：**在默认情况下，doxygen 会搜索具有典型 C/C++ 扩展名的文件，比如 *.c*、*.cc*、*.cpp*、*.h* 和 *.hpp*。如果 <FILE_PATTERNS> 标记没有相关联的值，doxygen 就会这样做。如果源代码文件采用不同的命名约定，就应该相应地更新这个标记。例如，如果项目使用 *.c86* 作为 C 文件扩展名，就应该在 <FILE_PATTERNS> 标记中添加这个扩展名。
- **<RECURSIVE>：**如果源代码层次结构是嵌套的，而且需要为所有层次上的 C/C++ 文件生成文档，就把这个标记设置为 **Yes**。例如，请考虑源代码根目录层次结构 /home/user1/project/kernel，其中有 /home/user1/project/kernel/vmm 和 /home/user1/project/kernel/asm 等子目录。如果这个标记设置为 *Yes*，doxygen 就会递归地搜索整个层次结构并提取信息。
- **<EXTRACT_ALL>：**这个标记告诉 doxygen，即使各个类或函数没有文档，也要提取信息。必须把这个标记设置为 **Yes**。
- **<EXTRACT_PRIVATE>：**把这个标记设置为 **Yes**。否则，文档不包含类的私有数据成员。
- **<EXTRACT_STATIC>：**把这个标记设置为 **Yes**。否则，文档不包含文件的静态成员（函数和变量）。

清单 3 给出一个 Doxyfile 示例。

清单 3. 包含用户提供的标记值的 doxyfile 示例

```
OUTPUT_DIRECTORY = /home/user1/docs
EXTRACT_ALL = yes
EXTRACT_PRIVATE = yes
EXTRACT_STATIC = yes
INPUT = /home/user1/project/kernel
#Do not add anything here unless you need to. Doxygen already covers all
#common formats like .c/.cc/.cxx/.c++/.cpp/.inl/.h/.hpp
FILE_PATTERNS =
RECURSIVE = yes
```

运行 doxygen

在 shell 提示下输入 `doxygen Doxyfile`（或者已为配置文件选择的其他文件名）运行 doxygen。在最终生成 Hypertext Markup Language（HTML）和 Latex 格式（默认）的文档之前，doxygen 会显示几个消息。在生成文档期间，在 <OUTPUT_DIRECTORY> 标记指定的文件夹中，会创建两个子文件夹 *html* 和 *latex*。清单 4 是一个 doxygen 运行日志示例。

清单 4. doxygen 的日志输出

```
Searching for include files...
Searching for example files...
Searching for images...
Searching for dot files...
Searching for files to exclude
Reading input files...
Reading and parsing tag files
Preprocessing /home/user1/project/kernel/kernel.h
...
Read 12489207 bytes
Parsing input...
Parsing file /project/user1/project/kernel/epico.cxx
...
Freeing input...
Building group list...
..
Generating docs for compound MemoryManager::ProcessSpec
...
Generating docs for namespace std
Generating group index...
Generating example index...
Generating file member index...
Generating namespace member index...
Generating page index...
Generating graph info page...
Generating search index...
Generating style sheet...
```

文档输出格式

除了 HTML 之外，doxygen 还可以生成几种输出格式的文档。可以让 doxygen 生成以下格式的文档：

- **UNIX 手册页：**把 <GENERATE_MAN> 标记设置为 **Yes**。在默认情况下，会在 <OUTPUT_DIRECTORY> 指定的目录中创建 *man* 子文件夹，生成的文档放在这

个文件夹中。必须把这个文件夹添加到 MANPATH 环境变量中。

- **Rich Text Format（RTF）：**把 <GENERATE_RTF> 标记设置为 **Yes**。把 <RTF_OUTPUT> 标记设置为希望放置 .rtf 文件的目录；在默认情况下，文档放在 OUTPUT_DIRECTORY 中的 *rtf* 子文件夹中。要想支持跨文档浏览，应该把 <RTF_HYPERLINKS> 标记设置为 **Yes**。如果设置这个标记，生成的 .rtf 文件会包含跨文档链接。
- **Latex：**在默认情况下，doxygen 生成 Latex 和 HTML 格式的文档。在默认的 Doxyfile 中，<GENERATE_LATEX> 标记设置为 **Yes**。另外，<LATEX_OUTPUT> 标记设置为 Latex，这意味着会在 OUTPUT_DIRECTORY 中创建 *latex* 子文件夹并在其中生成 Latex 文件。
- **Microsoft® Compiled HTML Help（CHM）格式：**把 <GENERATE_HTMLHELP> 标记设置为 **Yes**。因为在 UNIX 平台上不支持这种格式，doxygen 只在保存 HTML 文件的文件夹中生成一个 *index.hhp* 文件。您必须通过 HTML 帮助编译器把这个文件转换为 .chm 文件。
- **Extensible Markup Language（XML）格式：**把 <GENERATE_XML> 标记设置为 **Yes**。（注意，doxygen 开发团队还在开发 XML 输出）。

清单 5 提供的 Doxyfile 示例让 doxygen 生成所有格式的文档。

清单 5. 生成多种格式的文档的 Doxyfile

```
#for HTML
GENERATE_HTML = YES
HTML_FILE_EXTENSION = .htm

#for CHM files
GENERATE_HTMLHELP = YES

#for Latex output
GENERATE_LATEX = YES
LATEX_OUTPUT = latex

#for RTF
GENERATE_RTF = YES
RTF_OUTPUT = rtf
RTF_HYPERLINKS = YES

#for MAN pages
GENERATE_MAN = YES
MAN_OUTPUT = man
#for XML
GENERATE_XML = YES
```

---

doxygen 中的特殊标记

doxygen 包含几个特殊标记。

C/C++ 代码的预处理

为了提取信息，doxygen 必须对 C/C++ 代码进行预处理。但是，在默认情况下，它只进行部分预处理 —— 计算条件编译语句（#if…#endif），但是不执行宏展开。请考虑 清单 6 中的代码。

清单 6. 使用宏的 C++ 代码示例

```
#include <cstring>
#include <rope>

#define USE_ROPE

#ifdef USE_ROPE
  #define STRING std::rope
#else
  #define STRING std::string
#endif

static STRING name;
```

通过源代码中定义的 <USE_ROPE>，doxygen 生成的文档如下：

```
              Defines
    #define USE_ROPE
    #define STRING std::rope

Variables
    static STRING name
```

在这里可以看到 doxygen 执行了条件编译，但是没有对 STRING 执行宏展开。Doxyfile 中的 <ENABLE_PREPROCESSING> 标记在默认情况下设置为 *Yes*。为了执行宏展开，还应该把 <MACRO_EXPANSION> 标记设置为 **Yes**。这会使 doxygen 产生以下输出：

```
                Defines
   #define USE_ROPE
    #define STRING std::string

Variables
    static std::rope name
```

如果把 <ENABLE_PREPROCESSING> 标记设置为 *No*，前面源代码的 doxygen 输出就是：

```
                Variables
    static STRING name
```

注意，文档现在没有定义，而且不可能推导出 STRING 的类型。因此，总是应该把 <ENABLE_PREPROCESSING> 标记设置为 **Yes**。

在文档中，可能希望只展开特定的宏。为此，除了把 <ENABLE_PREPROCESSING> 和 <MACRO_EXPANSION> 标记设置为 **Yes** 之外，还必须把 <EXPAND_ONLY_PREDEF> 标记设置为 **Yes**（这个标记在默认情况下设置为 *No*），并在 <PREDEFINED> 或 <EXPAND_AS_DEFINED> 标记中提供宏的细节。请考虑 清单 7 中的代码，这里只希望展开宏 CONTAINER。

清单 7. 包含多个宏的 C++ 源代码

```
#ifdef USE_ROPE
  #define STRING std::rope
#else
  #define STRING std::string
#endif

#if ALLOW_RANDOM_ACCESS == 1
  #define CONTAINER std::vector
#else
  #define CONTAINER std::list
#endif

static STRING name;
static CONTAINER gList;
```

清单 8 给出配置文件。

清单 8. 允许有选择地展开宏的 Doxyfile

```
ENABLE_PREPROCESSING = YES
MACRO_EXPANSION = YES
EXPAND_ONLY_PREDEF = YES
EXPAND_AS_DEFINED = CONTAINER
...
```

下面的 doxygen 输出只展开了 CONTAINER：

```
                Defines
#define STRING   std::string
#define CONTAINER   std::list

Variables
static STRING name
static std::list gList
```

注意，只有 CONTAINER 宏被展开了。在 <MACRO_EXPANSION> 和 <EXPAND_ONLY_PREDEF> 都设置为 *Yes* 的情况下，<EXPAND_AS_DEFINED> 标记只选择展开等号操作符右边列出的宏。

对于预处理过程，要注意的最后一个标记是 <PREDEFINED>。就像用 -D 开关向 C++ 编译器传递预处理器定义一样，使用这个标记定义宏。请考虑 清单 9 中的 Doxyfile。

清单 9. 定义了宏展开标记的 Doxyfile

```
ENABLE_PREPROCESSING = YES
MACRO_EXPANSION = YES
EXPAND_ONLY_PREDEF = YES
EXPAND_AS_DEFINED =
```

```
PREDEFINED = USE_ROPE= \
                            ALLOW_RANDOM_ACCESS=1
```

下面是 doxygen 生成的输出：

```
                Defines
#define USE_CROPE
#define STRING   std::rope
#define CONTAINER   std::vector

Variables
static std::rope name
static std::vector gList
```

在使用 <PREDEFINED> 标记时，宏应该定义为 <*macro name*>=<*value*> 形式。如果不提供值，比如简单的 #define，那么只使用 <*macro name*>=<*spaces*> 即可。多个宏定义以空格或反斜杠（\）分隔。

从文档生成过程中排除特定文件或目录

在 Doxyfile 中的 <EXCLUDE> 标记中，添加不应该为其生成文档的文件或目录（以空格分隔）。因此，如果提供了源代码层次结构的根，并要跳过某些子目录，这将非常有用。例如，如果层次结构的根是 src_root，希望在文档生成过程中跳过 examples/ 和 test/memoryleaks 文件夹，Doxyfile 应该像 清单 10 这样。

清单 10. 使用 EXCLUDE 标记的 Doxyfile

```
INPUT = /home/user1/src_root
EXCLUDE = /home/user1/src_root/examples /home/user1/src_root/test/memoryleaks
...
```

生成图形和图表

在默认情况下，Doxyfile 把 <CLASS_DIAGRAMS> 标记设置为 **Yes**。这个标记用来生成类层次结构图。要想生成更好的视图，可以从 Graphviz 下载站点 下载 dot 工具。Doxyfile 中的以下标记用来生成图表：

- **<CLASS_DIAGRAMS>：**在 Doxyfile 中这个标记默认设置为 **Yes**。如果这个标记设置为 **No**，就不生成继承层次结构图。
- **<HAVE_DOT>：**如果这个标记设置为 **Yes**，doxygen 就使用 dot 工具生成更强大的图形，比如帮助理解类成员及其数据结构的协作图。注意，如果这个标记设置为 **Yes**，<CLASS_DIAGRAMS> 标记就无效了。
- **<CLASS_GRAPH>：**如果 <HAVE_DOT> 标记和这个标记同时设置为 **Yes**，就使用 dot 生成继承层次结构图，而且其外观比只使用 <CLASS_DIAGRAMS> 时更丰富。
- **<COLLABORATION_GRAPH>：**如果 <HAVE_DOT> 标记和这个标记同时设置为 **Yes**，doxygen 会生成协作图（还有继承图），显示各个类成员（即包含）及其继承层次结构。

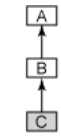
清单 11 提供一个使用一些数据结构的示例。注意，在配置文件中 <HAVE_DOT>、<CLASS_GRAPH> 和 <COLLABORATION_GRAPH> 标记都设置为 *Yes*。

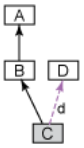
清单 11. C++ 类和结构示例

```
struct D {
  int d;
};

class A {
  int a;
};

class B : public A {
  int b;
};

class C : public B {
  int c;
  D d;
};
```

图 1 给出 doxygen 的输出。

图 1. 使用 dot 工具生成的类继承图和协作图





代码文档样式

到目前为止，我们都是使用 doxygen 从原本没有文档的代码中提取信息。但是，doxygen 也鼓励使用文档样式和语法，这有助于生成更详细的文档。本节讨论 doxygen 鼓励在 C/C++ 代码中使用的一些常用标记。更多信息参见 参考资料。

每个代码元素有两种描述：简短的和详细的。简短描述通常是单行的。函数和类方法还有第三种描述*体内描述*（*in-body description*），这种描述把在函数体中找到的所有注释块集中在一起。比较常用的一些 doxygen 标记和注释样式如下：

- **简短描述：**使用单行的 C++ 注释，或使用 <\brief> 标记。
- **详细描述：**使用 JavaDoc 式的注释 /** … test … */（注意开头的两个星号 [*]）或 Qt 式的注释 /*! … text … */。
- **体内描述：**类、结构、联合体和名称空间等 C++ 元素都有自己的标记，比如 <\class>、<\struct>、<\union> 和 <\namespace>。

为了为全局函数、变量和枚举类型生成文档，必须先对对应的文件使用 <\file> 标记。清单 12 给出的示例包含用于四种元素的标记：函数标记（<\fn>）、函数参数标记（<\param>）、变量名标记（<\var>）、用于 #define 的标记（<\def>）以及用来表示与一个代码片段相关的问题的标记（<\warning>）。

清单 12. 典型的 doxygen 标记及其使用方法

```
/*! \file globaldecls.h
      \brief Place to look for global variables, enums, functions
           and macro definitions
  */

/** \var const int fileSize
      \brief Default size of the file on disk
  */
const int fileSize = 1048576;

/** \def SHIFT(value, length)
      \brief Left shift value by length in bits
  */
#define SHIFT(value, length) ((value) << (length))

/** \fn bool check_for_io_errors(FILE* fp)
      \brief Checks if a file is corrupted or not
      \param fp Pointer to an already opened file
      \warning Not thread safe!
  */
bool check_for_io_errors(FILE* fp);
```

下面是生成的文档：

```
                Defines
#define SHIFT(value, length)   ((value) << (length))
             Left shift value by length in bits.

Functions
bool check_for_io_errors (FILE *fp)
        Checks if a file is corrupted or not.

Variables
const int fileSize = 1048576;
Function Documentation
bool check_for_io_errors (FILE* fp)
Checks if a file is corrupted or not.

Parameters
              fp: Pointer to an already opened file

Warning
Not thread safe!
```

结束语

本文讨论如何用 doxygen 从遗留的 C/C++ 代码提取出大量相关信息。如果用 doxygen 标记生成代码文档，doxygen 会以容易阅读的格式生成输出。只要以适当的方式使用，doxygen 就可以帮助任何开发人员维护和管理遗留系统。

参考资料

学习

- 您可以参阅本文在 developerWorks 全球站点上的 英文原文 。
- doxygen 站点 包含关于 doxygen 的非常有价值的手册和几篇文章。
- 下载 dot 实用程序。
- AIX and UNIX 专区：developerWorks 的“AIX and UNIX 专区”提供了大量与 AIX 系统管理的所有方面相关的信息，您可以利用它们来扩展自己的 UNIX 技能。
- AIX and UNIX 新手入门：访问“AIX and UNIX 新手入门”页面可了解更多关于 AIX 和 UNIX 的内容。
- AIX and UNIX 专题汇总：AIX and UNIX 专区已经为您推出了很多的技术专题，为您总结了很多热门的知识点。我们在后面还会继续推出很多相关的热门专题给您，为了方便您的访问，我们在这里为您把本专区的所有专题进行汇总，让您更方便的找到您需要的内容。
- developerWorks 技术活动和网络广播：随时关注 developerWorks 技术活动和网络广播。
- Podcasts：收听 IBM 技术专家的访谈录。

获得产品和技术

- 下载 doxygen。
- IBM 试用软件：使用可从 developerWorks 直接下载的软件构建您的下一个开发项目。

讨论

-
  - AIX Forum
  - AIX Forum for Developers
  - Cluster Systems Management
  - IBM Support Assistant Forum
  - Performance Tools Forum
  - Virtualization Forum
  - 更多 AIX 和 UNIX 论坛

关于作者

Arpan Sen 是一位资深工程师，从事电子设计自动化行业的软件开发。他花了好几年研究多种风格的 UNIX，包括 Solaris、SunOS、HP-UX 和 IRIX，以及 Linux 和 Microsoft Windows。他主要对软件性能优化技术、图形理论和并行计算感兴趣。Arpan 拥有软件系统研究生学位。

关闭 [x]

请选择您的昵称：

当您初次登录到 developerWorks 时，将会为您创建一份概要信息，您需要指定一个昵称。您的昵称将和您在 developerWorks 发布的内容显示在一起。

昵称长度在 3 至 31 个字符之间。 您的昵称在 developerWorks 社区中必须是唯一的，并且出于隐私保护的原因，不能是您的电子邮件地址。

昵称： （长度在 3 至 31 个字符之间）

单击**提交**则表示您同意developerWorks 的条款和条件。 使用条款.

提交 取消

所有提交的信息确保安全。

平均分 (11个评分)

1 星

1 星
2 星

2 星
3 星

3 星
4 星

4 星
5 星

5 星

提交

添加评论:

请 登录 或 注册 后发表评论。

注意：评论中不支持 HTML 语法

有新评论时提醒我剩余 1000 字符

发布

快来添加第一条评论