# 基于Doxygen的代码注释规范

## 一、Doxygen系列软件介绍

### 1、Doxygen

Doxygen是一种开源跨平台的，以类似JavaDoc风格描述的文档系统，完全支持C、C++、Java、[Objective-C](http://baike.baidu.com/view/459423.htm" \t "_blank)和IDL语言，部分支持PHP、C#。注释的语法与Qt-Doc、KDoc和JavaDoc兼容。Doxgen可以从一套归档[源文件](http://baike.baidu.com/view/385166.htm" \t "_blank)开始，生成HTML格式的在线类[浏览器](http://baike.baidu.com/view/7718.htm" \t "_blank)，或离线的LATEX、RTF参考手册。

Doxygen能将程序中的特定批注转换成为说明文件。它可以依据程序本身的结构，将程序中按规范注释的批注经过处理生成一个纯粹的参考手册，通过提取代码结构或借助自动生成的包含依赖图（include dependency graphs）、继承图（inheritance diagram）以及协作图（collaboration diagram）来可视化文档之间的关系， Doxygen生成的帮助文档的格式可以是CHM、RTF、PostScript、PDF、HTML等。

### 2、graphviz

Graphviz(Graph Visualization Software)是一个由AT&T实验室启动的开源工具包,用于绘制DOT语言脚本描述的图形。

要使用Doxygen生成依赖图、继承图以及协作图，必须先安装graphviz软件。

### 3、HTML Help WorkShop

微软出品的HTML Help WorkShop是制作CHM文件的最佳工具，它能将HTML文件编译生成CHM文档。

Doxygen软件默认生成HTML文件或Latex文件，我们要通过HTML生成CHM文档，需要先安装HTML Help WorkShop软件，并在Doxygen中进行关联。

## 二、软件下载与安装

**Doxygen**下载（doxygen-1.8.7-setup.exe）：

<http://www.stack.nl/~dimitri/doxygen/download.html>

**graphviz**（for windows）下载：

<http://www.graphviz.org/Download..php>

**HTML Help WorkShop**（1.32）下载：

<http://download.microsoft.com/download/0/a/9/0a939ef6-e31c-430f-a3df-dfae7960d564/htmlhelp.exe>

软件安装都选择默认方式，点击下一步直至安装完成。

安装完后进行Doxygen配置时需要关联graphviz和HTML Help WorkShop的安装路径。

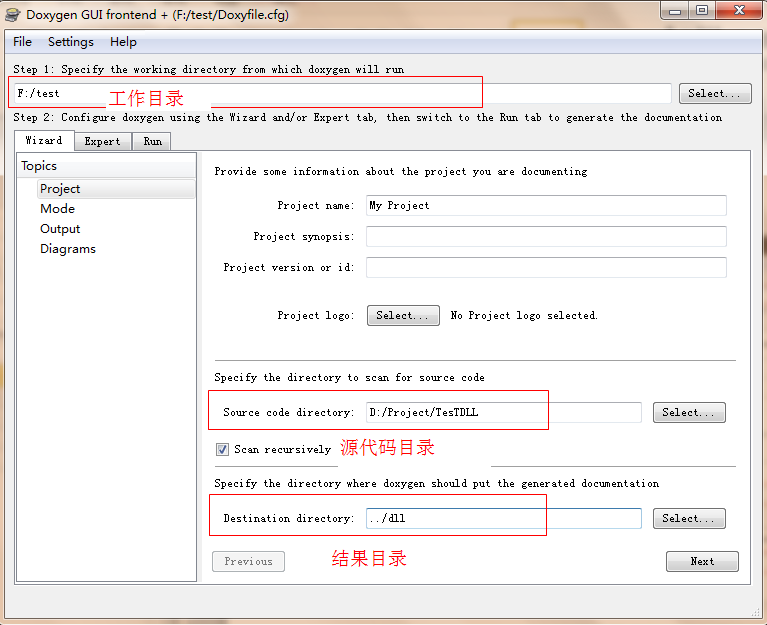
## 三、软件配置使用

我们的所有配置都在Doxywizard中进行，生成参考手册是通过运行Doxywizard得到。

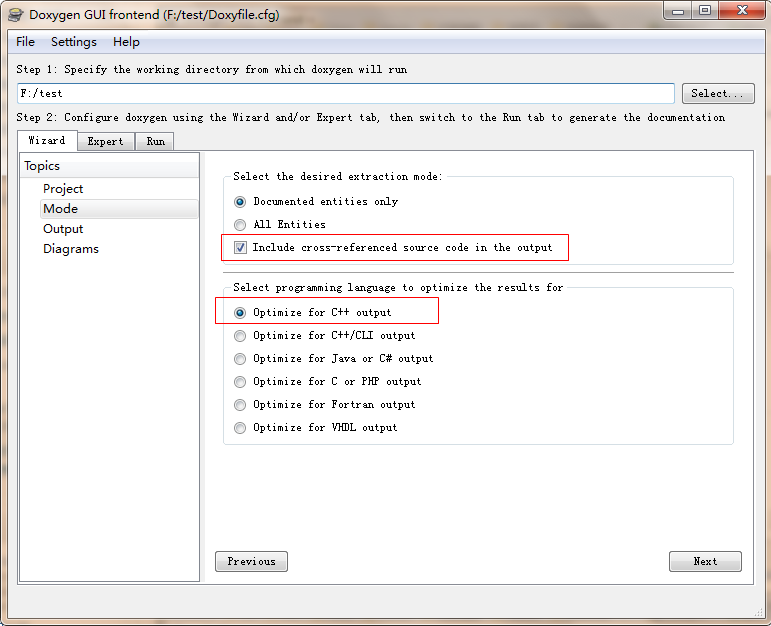
（1）Wizard->Project最重要的是工作目录，源代码目录，生成参考文件目录三处的设定，其它项目名称、项目简介、版本和标识可以依照实际情况选填。

工作目录是新建的一个目录，在配置完成之后可以把配置文件存在这个目录里，每次从这个目录中导入配置文件（.cfg），然后进行说明文档生成。

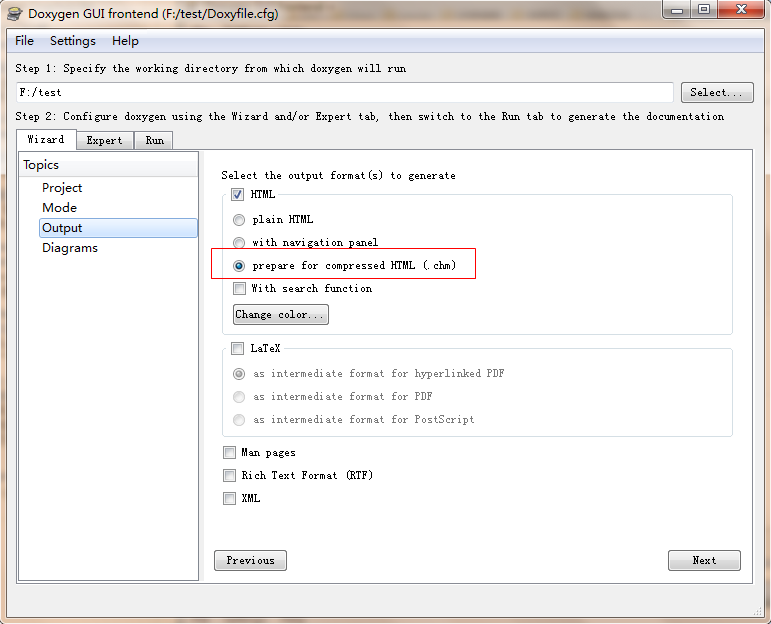
源代码目录和最终的结果目录在每一次运行Doxywizard时都进行设定。

****

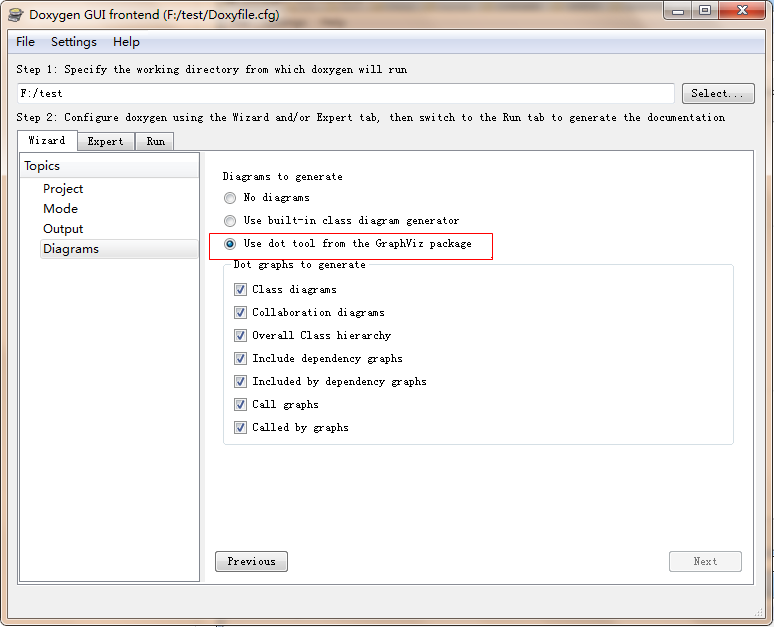
（2）Wizard->Mode 选择编程语言对应的最优化结果，按照编程语言选择。

****

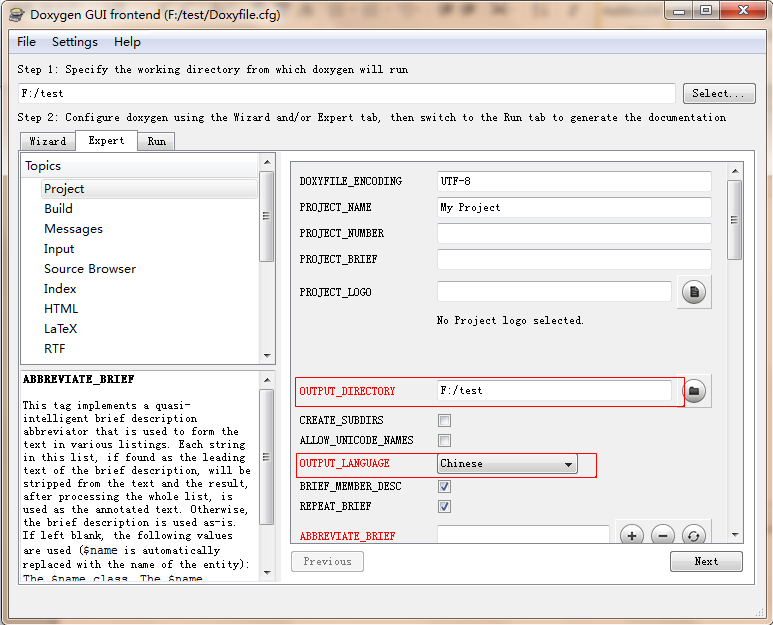
（3）Wizard->Output 选择输出格式，选HTML下的（.chm）项，为最后生成chm做准备。由于不需要LaTeX结果，不选此项。



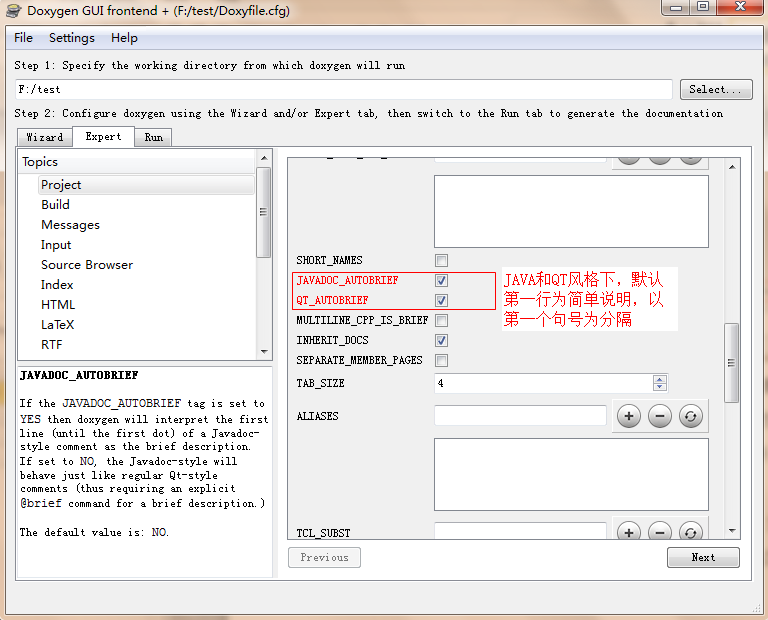
（4）Wizard->Diagrams 选择dot tool项，通过GraphViz来作图。



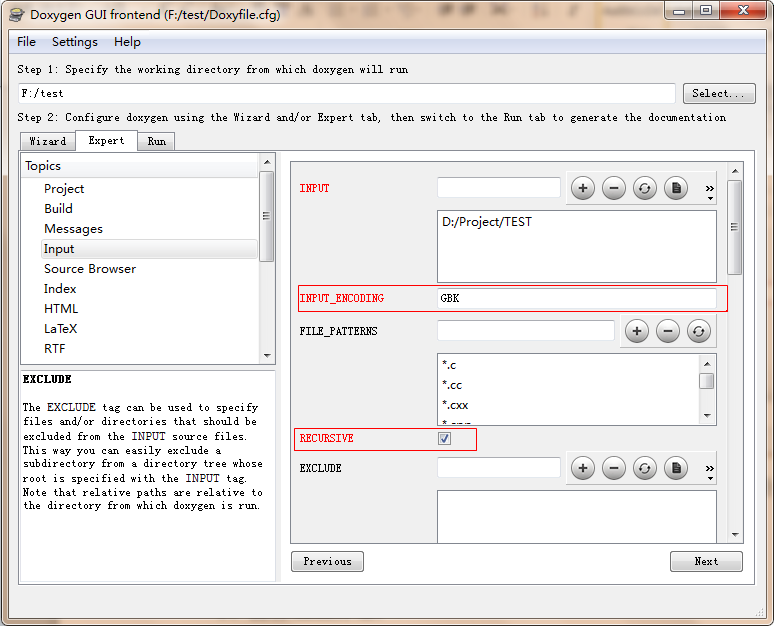
（5）Expert-> Project 选择输出目录，选着输出语言。如果代码中采用了中文注释，此处选择为中文。



向下拉滑条，看见有JAVADOC\_AUTOBRIEF和QT\_AUTOBRIEF两个框，如果勾选了，在这两种风格下默认第一行为简单说明，以第一个句号为分隔；如果不选，则需要按照Doxygen的指令@brief来进行标准注释。

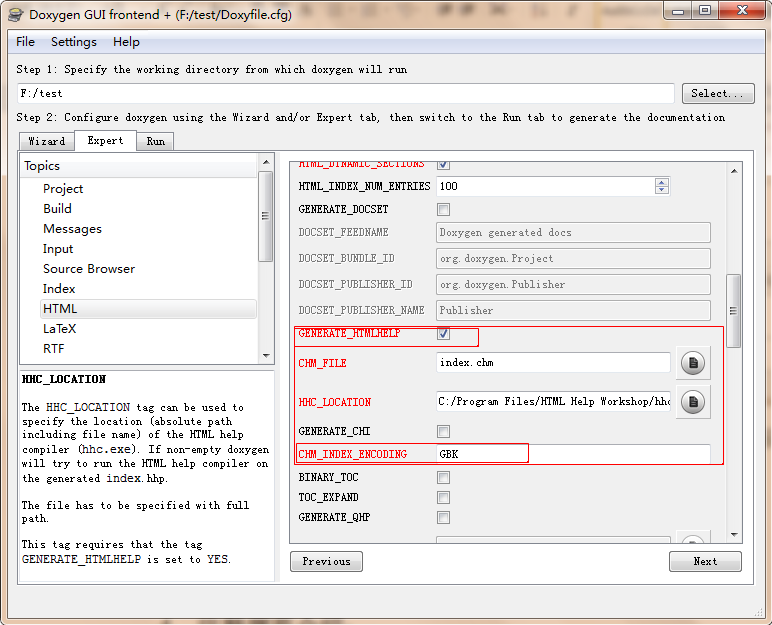


（6）Expert-> Input 将输入编码方式改为GBK方式，确保输出中不会由于UTF-8方式导致乱码。

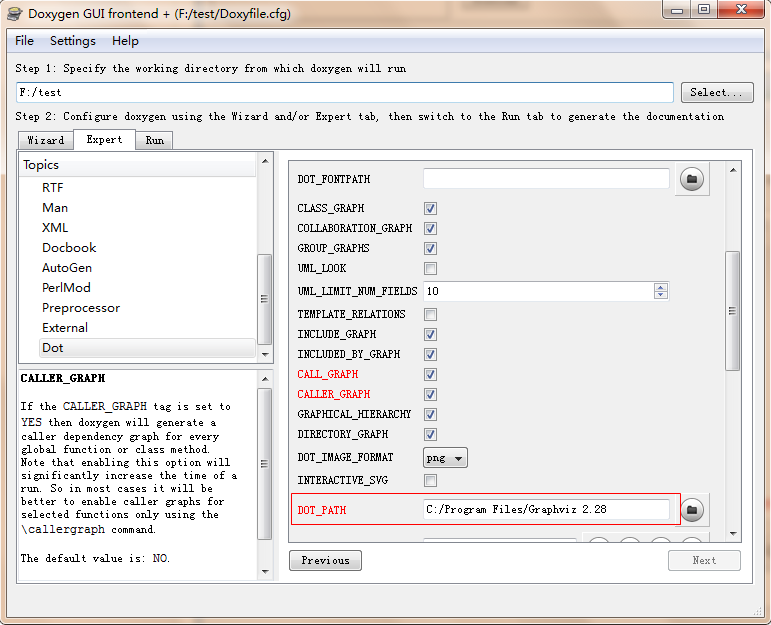


**最后也是经常遇到的问题就是DoxyGen生成的CHM文件的左边树目录的中文变成了乱码**。这个 只需要将chm索引的编码类型修改为GB2312即可。**在HTML的CHM\_INDEX\_ENCODING中输入GB2312即可。**

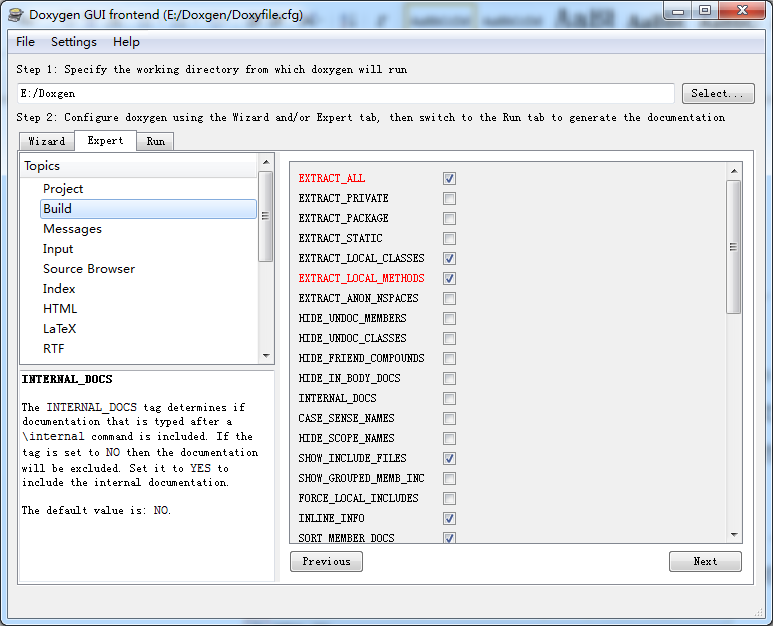
（7）Expert-> HTML 勾选生成HTMLHELP项，输入生成CHM名称，在HHC\_LOCATION中填入HTMLHELP WORKSHOP安装目录中hhc.exe的路径，将chm编码方式改为GBK方式，与第（6）步中的输入编码方式一致。



（8）Expert->Dot 在Dot\_PATH中填写GraphViz的安装路径。



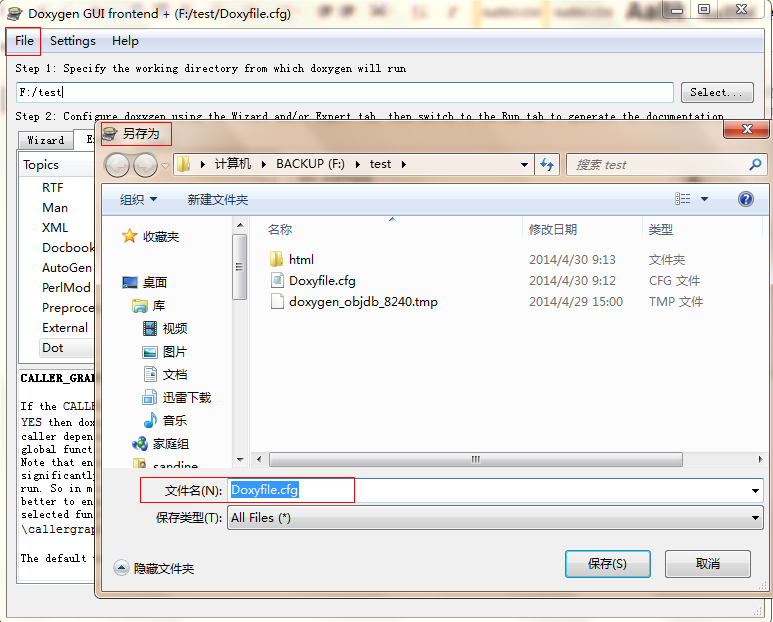
**需要在build中配置EXTRACT\_ALL和LOCAL\_METHODS才能生成所有的变量和函数。**



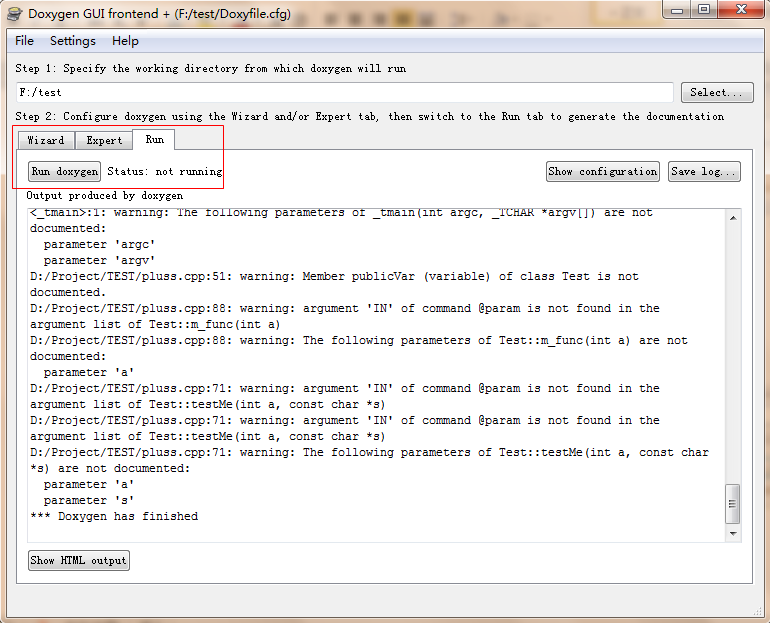
（9）存储配置信息。到上一步Doxygen已经完全配置好，可以在Run中点击运行了，但为了保存以上配置信息，可以将配置好的文件存一个.cfg文件，之后再运行Doxygen时只需要将该文件用Doxygen打开，改变第（1）步中的输入、输出目录及工程的信息再运行。

File->Save as, 取一个名，默认为Doxyfile，加.cfg,点击保存。

如果需要改变配置文件，改动之后再Save替换之前的配置文件即可。



（10）Run->Run Doxygen 即可运行Doxygen，运行完成后在输出目录中的html文件夹中找到index.chm文件即为输入代码的文档说明。



## 四、Doxygen注释规范简介

### 1、Doxygen规范综述

简要的说，Doxygen注释块其实就是在C、C++注释块的基础添加一些额外标识,使Doxygen把它识别出来, 并将它组织到生成的文档中去。

在每个代码项中都可以有两类描述：一种就是brief描述,另一种就是detailed。两种都是可选的，但不能同时没有。简述(brief)就是在一行内简述地描述。而详细描述(detailed)则提供更长,更详细的文档。

在Doxygen中,主要通过以下方法将注释块标识成详细(detailed)描述:

**JavaDoc**风格，在C风格注释块开始使用两个星号'\*'：

/\*\*

\* ... 描述 ...

\*/

**Qt**风格代码注释,即在C风格注释块开始处添加一个叹号'!':

/\*!

\* ... 描述 ...

\*/

使用连续两个以上**C++**注释行所组成的注释块, 而每个注释行开始处要多写一个斜杠或写一个叹号：

///

/// ... 描述 ...

///

同样的，简要说明（brief）有也有多种方式标识，这里推荐使用@brief命令强制说明，例如：

/\*\*

\* @brief 简要注释Brief Description.

\*/

或者

///

/// @brief 简要注释Brief Description.

///

**需要注意以下几点：**

（1）Doxygen并不处理所有的注释，doxygen重点关注与程序结构有关的注释，比如：文件、类、结构、函数、全局变量、宏等注释，而忽略函数内**局部变量、代码**等的注释。

（2）注释应写在对应的函数或变量前面。JavaDoc风格下，自动会把第一个句号"."前的文本作为简要注释，后面的为详细注释。你也可以用空行把简要注释和详细注释分开。注意要设置JAVADOC\_AUTOBRIEF或者QT\_AUTOBRIEF设为YES。

（3）先从文件开始注释，然后是所在文件的全局函数、结构体、枚举变量、命名空间→命名空间中的类→成员函数和成员变量。

（4）Doxygen无法为DLL中定义的类导出文档

例如：

class \_\_declspec(dllexport) CClassName:public CObject

{

}

所生成的文档中发现Doxygen无法识别出DLL中定义的类。

### 2、Doxygen常用指令

如前面所述的@brief命令，还可以在注释中加入其他Doxygen支持的指令，控制输出文档的排版格式，使用这些指令时需要在前面加上“\”或者“@”（JavaDoc风格）符号，告诉Doxygen这些是一些特殊的指令，通过加入这些指令以及配备相应的文字，可以生成更加丰富的文档

下表列出常用的Doxygen指令：

|  |  |
| --- | --- |
| @file | 档案的批注说明。 |
| @author | 作者的信息 |
| @brief | 用于class 或function的简易说明  eg：  @brief 本函数负责打印错误信息串 |
| @param | 主要用于函数说明中，后面接参数的名字，然后再接关于该参数的说明 |
| @return | 描述该函数的返回值情况  eg:  @return 本函数返回执行结果，若成功则返回TRUE，否则返回FLASE |
| @retval | 描述返回值类型  eg:  @retval NULL 空字符串。 @retval !NULL 非空字符串。 |
| @note | 注解 |
| @attention | 注意 |
| @warning | 警告信息 |
| @enum | 引用了某个枚举，Doxygen会在该枚举处产生一个链接  eg：  @enum CTest::MyEnum |
| @var | 引用了某个变量，Doxygen会在该枚举处产生一个链接  eg：  @var CTest::m\_FileKey |
| @class | 引用某个类，  格式：@class <name> [<header-file>] [<header-name>]  eg:  @class CTest "inc/class.h" |
| @exception | 可能产生的异常描述  eg:  @exception 本函数执行可能会产生超出范围的异常 |

**Modules（模块）：**

Modules是一种归组things在分离的page上的方式。组的成员可以是file，namespace，classes，functions，variables，enums，typedefs和defines，但也可以是其他groups。

要定义一个group，应该在一个特殊注释块放置\defgroup。命令的第一个参数应该是唯一标志该group的标签。要将一个entity归为某个group的一个member，在entity前放置\ingroup命令。第二个参数是group的title。

要避免在注释中每个member前放置\ingroup命令，可以将member用@{和@}封装起来。@{@}标记可以放置group的注释中，也可以在一个独立的注释块

使用这些group的标记符号groups也可以嵌套。

如果多次使用一个group标签，将会出错。如果不希望doxygen强行执行唯一标签，可以使用\addtogroup而非\defgroup。运作方式和\defgroup很像，但是如果该group已经定义，它默认向已存在的注释中添加一个新的项。Group的title对此命令是可选的，也可以考虑使用它。

/\*

\* @defgroup 模块名 模块的说明文字

\* @{

\*/

... 定义的内容 ...

/\*\* @} \*/

// 模块结尾这样可以在其他地方以更加详细的说明添加members到一个group。

注意compound entities（例如classes，files和namespaces）可以放在多个groups中，但是members（例如variables，functions，typedefs和enmus）只可以归于一个group

Member Groups

如果一个compound（例如一个class或file）有多个members，通常我们希望将其group。Doxygen已经可以自动按照类型和protection级别将这些things归组在一起，但可能你会认为仅仅这样是不够的或者这种缺省的方法是错误的。例如你认为有不同（语法）的类型需要归入同一个group（语意）。

这样定义一个member group：

//@{

...

//@}

块或者使用

/\*@{\*/

...

/\*@}\*/

### 3、Doxygen注释实例

下述实例均以JavaDoc风格进行注释，在日常规范中也推荐尽量使用JavaDoc风格的注释。

1. **文件注释**

文件不在任何东西里面，所以不能像类、函数等在上方放注释，只能用@file方式定义，其格式如下：

/\*\* @file [file‐name]

\* @brief brief description

\* @author <list of authors>

\* [@author <authors description>]

\* @date <date>

\* @version <version number>

\* @note

\* detailed description

\*/

[]表示可选项，{}表示重复0到N次，<>表示必须参数  
 一般@file后我们空着，Doxygen会默认为是@file所在文件的文件名。

@brief 为简明注释

@aothor 为作者列表

@date 为日期

@version 为版本号

@note 为注解

一个文件注释实例：

1. **类和成员注释**

/\*\*

\* @class <class‐name> [header‐file] [<header‐name]

\* @brief brief description

\* @author <list of authors>

\* @note

\* detailed description

\*/

如果对文件、结构体、联合体、类或者枚举的成员进行文档注释的话,并且要在成员中间添加注释,而这些注释往往都是在每个成员后面。为此,可以使用在注释段中使用'<'标识。

int var; /\*\*< Detailed description after the member \*/

对一个类的注释例子如下：

class Test

{

public:

/\*\* @brief A enum, with inline docs \*/

enum TEnum

{

TVal1, /\*\*< enum value TVal1. \*/

TVal2, /\*\*< enum value TVal2. \*/

TVal3 /\*\*< enum value TVal3. \*/

}

\*enumPtr, /\*\*< enum pointer. \*/

enumVar; /\*\*< enum variable. \*/

/\*\* @brief A constructor. \*/

Test();

/\*\* @brief A destructor. \*/

~Test();

/\*\* @brief a normal member taking two arguments and returning an integer value. \*/

int testMe(int a,const char \*s);

/\*\* @brief A pure virtual member.

\* @param[in] c1 the first argument.

\* @param[in] c2 the second argument.

\* @see testMe()

\*/

virtual void testMeToo(char c1,char c2) = 0;

int publicVar;/\*\*< a public variable. \*/

/\*\* @brief a function variable, note Details. \*/

int (\*handler)(int a,int b);

/\*\* @brief brief before delaration \*/

int m\_func(int a);

};

1. **函数注释**

任何函数都必须要有简要注释和详细注释，习惯用法如下：

/\*\*

\* @brief brief description

\* @author <list of authors>

\* @param[in|out] <parameter‐name> <parameter description>

\* @exception <exception‐object> <exception description>

\* @return <description of the return value>

\* @note

\* detailed description

\* @remarks <remark text>

\*/

举例说明：

/\*\*

\* @breif下面是一个含有两个参数的函数的注释说明（简述）

\*

\* 这里写该函数的详述信息

\* @param[in] a 被测试的变量（param描述参数）

\* @param[in] s 指向描述测试信息的字符串

\* @return 测试结果 （return描述返回值）

\* @see Test()（本函数参考其它的相关的函数，这里作一个链接）

\* @note (note描述需要注意的问题)

\*/

int testMe(int a,const char \*s)

1. **枚举注释**

每个枚举定义必须添加注释,格式如下:

/\*\* Another enum, with inline docs \*/

enum AnotherEnum

{

V1, /\*\*< value 1 \*/

V2 /\*\*< value 2 \*/

};

1. **全局变量和全局宏**

全局变量和全局宏必须要有注释，

如果注释较短，则可以在

所注释代码上方用 /\*\* @brief some brief description \*/

或右方用 /\*\*< some brief description \*/

进行简要注释。

举例如下

/\*\*

/\*\*

\* @brief这是一个全局变量。

\*

\* doxygen 默认的注释都是写在要注释的部分的前面，

\* 写在后面的注释要加一个额外的符号'<'。

\* 后面的注释往往用来说明一个class,struct, union, or enum 的成员变量

\*/

int n;

int m; /\*\*< 这是另一个全局变量。写在后面的注释也可采用"/\*\*< ...\*/"的方式 \*/

1. **Modules（模块注释）**

Group定义命令的优先级（从高到低）：[\ingroup](#cmdingroup)，[\defgroup](#cmddefgroup)，[\addtogroup](#cmdaddtogroup)，[\weakgroup](#cmdweakgroup)。而[\weakgroup](#cmdweakgroup)很像一个有低优先级的[\addtogroup](#cmdaddtogroup)。它被设计为实现一个“lazy”的group定义方法：可以在.h文件中使用高优先级来定义结构，在.cpp文件中使用[\weakgroup](#cmdweakgroup)这样不会重复.h文件中的层次结构。

/\*\*

\* \ingroup A

\*/

extern int VarInA;

/\*\*

\* \defgroup IntVariables Global integer variables

\*/

/\*@{\*/

/\*\* an integer variable \*/

extern int IntegerVariable;

/\*@}\*/

/\*\*

\* \defgroup Variables Global variables

\*/

/\*@{\*/

/\*\* a variable in group A \*/

int VarInA;

int IntegerVariable;

/\*@}\*/

如果你更喜欢C style注释。需要注意的是所有的members必须写在其中。

在//@{之前还可以加一个注释块，这个注释块应该包含@name（或者\name）来指明group的header。可选的，这个注释块可以包含group的更详细的信息。

Member groups不允许使用嵌套。

如果一个类中的某个member group中所有的members有相同的type和protection level（例如都是static public members），那么这整个都会作为该type/protection level group的subgroup显式出来（例如，这个group作为“static public members”section的subsection）。如果两个或更多成员有不同的类型，那么这个group会和自动产生的groups放在同一个level。

Example:

/\*\* A class. Details \*/

class Test

{

public:

//@{

/\*\* Same documentation for both members. Details \*/

void func1InGroup1();

void func2InGroup1();

//@}

/\*\* Function without group. Details. \*/

void ungroupedFunction();

void func1InGroup2();

protected:

void func2InGroup2();

};

void Test::func1InGroup1() {}

void Test::func2InGroup1() {}

/\*\* @name Group2

\* Description of group 2.

\*/

//@{

/\*\* Function 2 in group 2. Details. \*/

void Test::func2InGroup2() {}

/\*\* Function 1 in group 2. Details. \*/

void Test::func1InGroup2() {}

//@}

/\*! \file

\* docs for this file

\*/

//@{

//! one description for all members of this group

//! (because DISTRIBUTE\_GROUP\_DOC is YES in the config file)

#define A 1

#define B 2

void glob\_func();

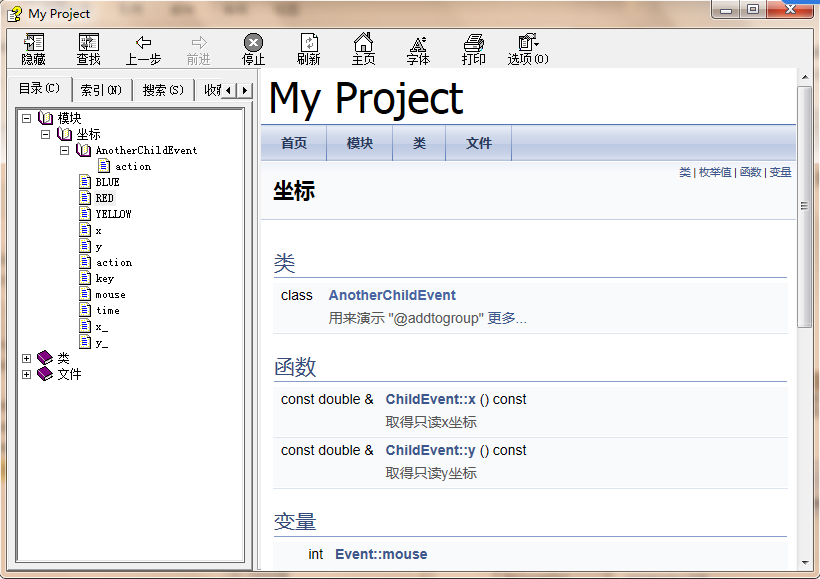
//@}

下面给出一个例子程序生成说明文档chm的对照示意图：

该例子包括一个doxytest.h和doxytest.cpp，大部分的声明、定义、注释都写在doxytest.h内。最后生成的说明文档包括三大类，模块、类和文件。

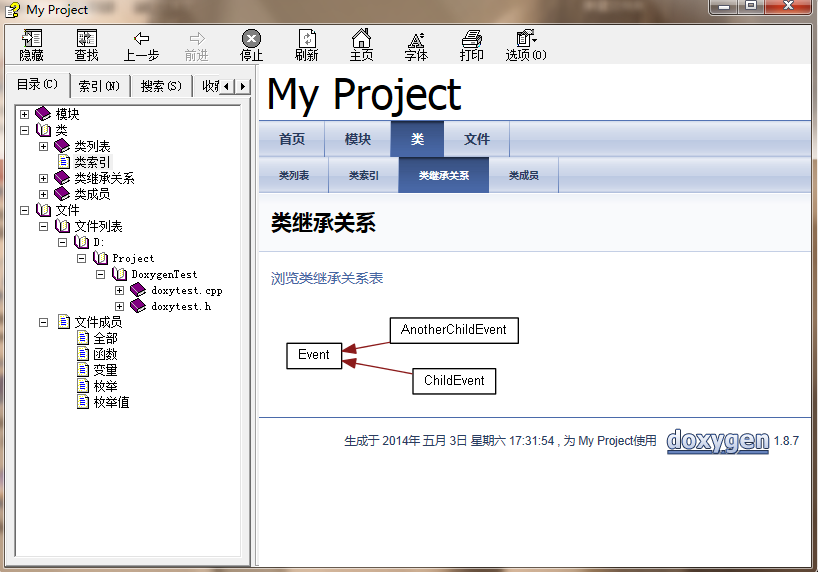
**模块：**

根据项目需要将一组成员（包括类，函数，变量）建成一个group，在说明文档中会以模块的形式显示出来。

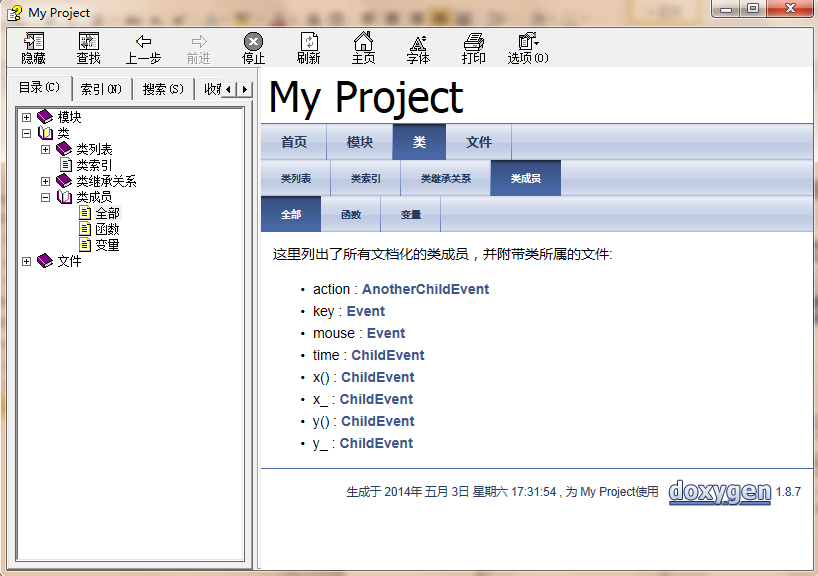


**类：**

类继承关系图：



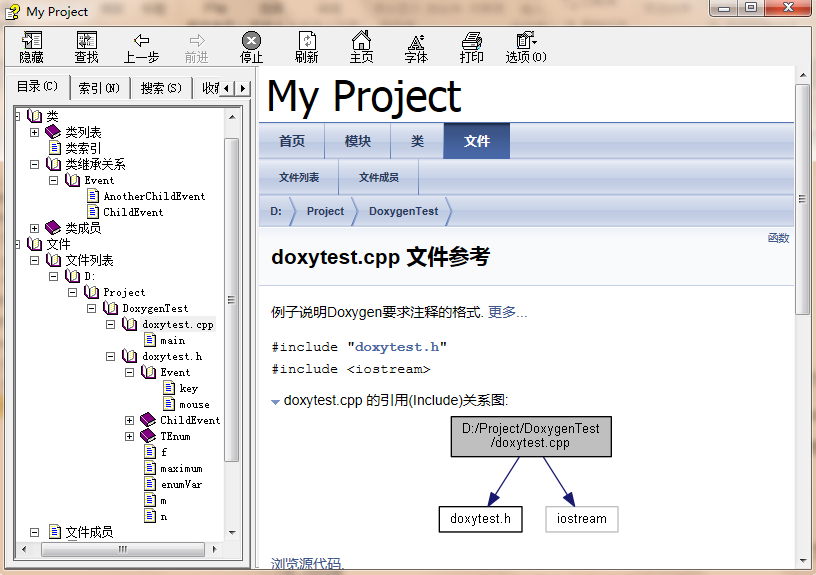
类成员，可以分为成员函数和成员变量进行查看



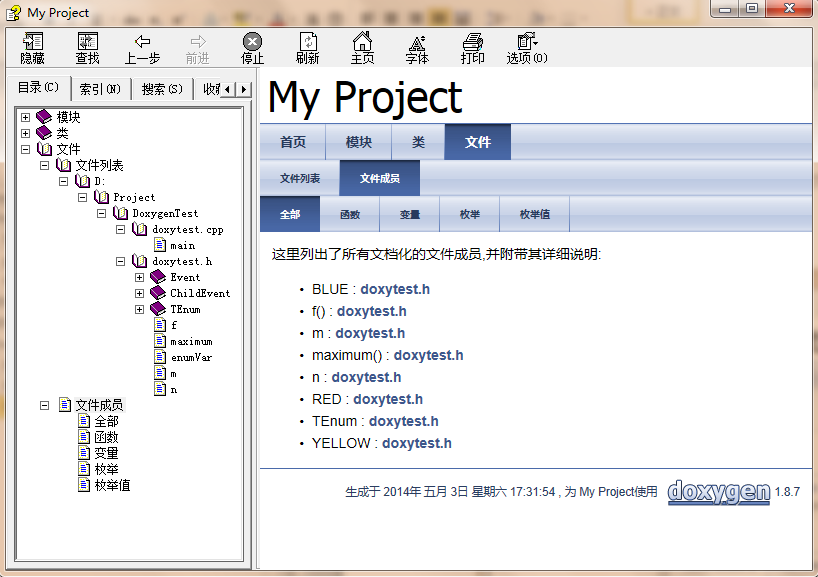
**文件：**

doxytest.cpp中只有main函数，其他声明与定义等放在doxytest.h中。

文件引用关系示意图：



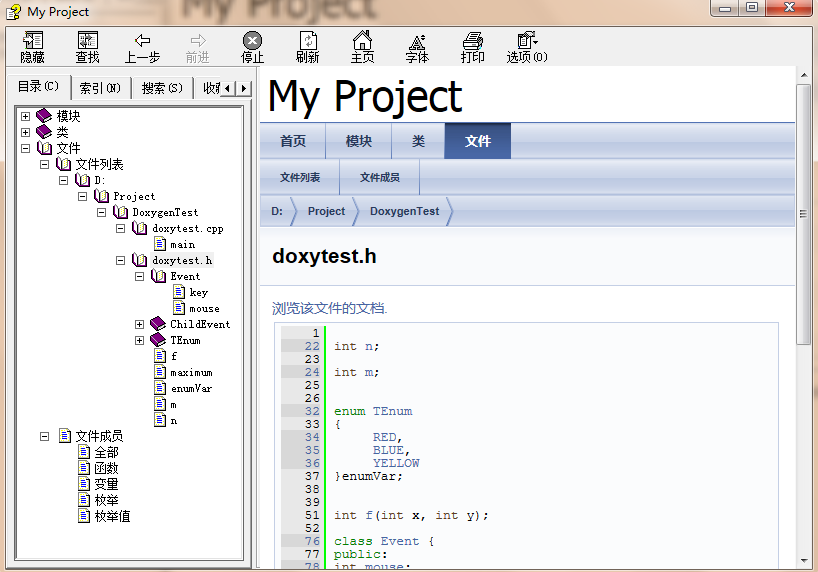
文件成员列表图：



可以通过文件列表的方法来查看文件中的各项成员，也可以通过不同文件成员进行查看。



说明文档中还可以查看源代码，doxygen识别并处理过的注释不显示，其余注释会显示出来。



附加问题：

**1.如何隐藏源码？**

开发者不愿意公布源码而只愿意公布接口和注释、调用信息。

**SOURCE\_BROWSER NO 如果为YES，源代码文件会被包含在文档中**

**INLINE\_SOURCE 也是用于控制源码选项**

**v1.87单独在Expert中开辟了这一栏目。**

