



个人资料



hbaizj

访问: 60343次

积分: 7

等级: **BLOG > 1**

排名: 千里之外

原创: 35篇 转载: 29篇

译文: 0篇 评论: 23条

文章搜索

文章分类

[java](#) (2)

[VC](#) (21)

[Windows Mobile](#) (6)

[STM32](#) (15)

文章存档

[2014年12月](#) (1)

[2016软考项目经理实战班](#) [学院周年礼-顶尖课程钜惠呈现](#) [有奖征文: 在云上开发的无限可能](#) [【博客专家】有奖试读—Windows PowerShell实战指南](#)

## C||C++中几个罕见却有用的预编译和宏定义

标签: [c++](#) [c](#) [math.h](#) [编译器](#) [mfc](#) [preprocessor](#)

2008-11-10 14:11

743人阅读

[评论\(0\)](#)

[收藏](#)

[举报](#)

版权声明: 本文为博主原创文章, 未经博主允许不得转载。

### 1: #error

语法格式如下:

#error token-sequence

其主要的作用是在编译的时候输出编译错误信息token-sequence, 从方便程序员检查程序中出现的错误。例如下面的程序

```
#include "stdio.h"
```

```
int main(int argc, char* argv[])
```

```
{
```

```
#define CONST_NAME1 "CONST_NAME1"
```

```
printf("%s/n",CONST_NAME1);
```

```
#undef CONST_NAME1
```

```
#ifndef CONST_NAME1
```

```
#error No defined Constant Symbol CONST_NAME1
```

```
#endif
```

```
{
```

2014年11月 (2)  
2014年04月 (4)  
2013年12月 (2)  
2013年07月 (1)

展开

阅读排行

- 以及上拉输入、下拉输入 (5008)
- FreeType2教程 (3549)
- STM32精确延时(非中断, (3182)
- freetype2教程(转) (3149)
- 全面了解setjmp与longjmp (3139)
- 使用FreeType实现矢量字 (2280)
- 像素与毫米的转换 (2024)
- 也谈C++中char\*与wchar\_t (1748)
- VC 2005 对于CString和CFile (1501)
- 《FreeType Glyph Conv (1246)

评论排行

- 全面了解setjmp与longjmp (5)
- 也谈C++中char\*与wchar\_t (4)
- 以及上拉输入、下拉输入 (3)
- VC中#pragma的使用方 (2)
- VC 2005 对于CString和CFile (2)
- freetype2教程(转) (1)
- FreeType2教程 (1)
- 《FreeType Glyph Conv (1)
- 修改WinCE启动界面（笔 (1)
- static\_cast、dynamic\_cast (1)

推荐文章

- \*算法与数据结构学习资源大搜罗——良心推荐
- \*架构设计：系统间通信（17）——服务治理与Dubbo 中篇（分析）
- \*数据库性能优化之SQL语句优化
- \*Android应用开发allowBackup

```
#define CONST_NAME2 "CONST_NAME2"

printf("%s/n",CONST_NAME2);

}

printf("%s/n",CONST_NAME2);

return 0;

}
```

在编译的时候输出如编译信息

fatal error C1189: #error : No defined Constant Symbol CONST\_NAME1

## 2: # pragma

其语法格式如下:

### # pragma token-sequence

此指令的作用是触发所定义的动作。如果token-sequence存在，则触发相应的动作，否则忽略。此指令一般为编译系统所使用。例如在Visual C++.Net 中利用# pragma once 防止同一代码被包含多次。

## 3: #line

此命令主要是为强制编译器按指定的行号，开始对源程序的代码重新编号，在调试的时候，可以按此规定输出错误代码的准确位置。

形式1

语法格式如下:

### # line constant “filename”

其作用是使得其后的源代码从指定的行号constant重新开始编号，并将当前文件的名命名为filename。例如下面的程序如下:

```
#include "stdio.h"

void Test();

#line 10 "Hello.c"

int main(int argc, char* argv[])

{

#define CONST_NAME1 "CONST_NAME1"

printf("%s/n",CONST_NAME1);

#undef CONST_NAME1

printf("%s/n",CONST_NAME1);

{

#define CONST_NAME2 "CONST_NAME2"

printf("%s/n",CONST_NAME2);

}
```

敏感信息泄露的一点反思

[\\*Linux多线程实践（四）线程的特定数据](#)

[\\*深度学习2015年文章整理（CVPR2015）](#)

最新评论

[也谈C++中char\\*与wchar\\_t\\*之间](#)  
[zuo\\_8267225](#): 我试了有问题

[全面了解setjmp与longjmp\(C语言](#)  
[barry\\_di](#): 很详细，不错。

[全面了解setjmp与longjmp\(C语言](#)  
[eziowayne](#): 第二个例子完全不能理解啊，if(1) if(2)这些有什么用呢？？

[STM32精确延时\(非中断,非ST库i](#)  
[wgwork](#): 理论上任何一种定时器都可以用这种方法来延时，不一定非得用systick.

[以及上拉输入、下拉输入、浮空转](#)  
[u010243385](#): 很实用

[以及上拉输入、下拉输入、浮空转](#)  
[sun\\_z\\_x](#): 好厉害

[以及上拉输入、下拉输入、浮空转](#)  
[lby1168](#): 很详细，谢谢

[全面了解setjmp与longjmp\(C语言](#)  
[小菩提的尾巴](#): 不错，谢了，收藏去

[FreeType2教程](#)  
[Max\\_\\_](#): 这教程讲得很全面。。

[也谈C++中char\\*与wchar\\_t\\*之间](#)  
[penglijiang](#): 谢谢你 不错

```
printf("%s/n",CONST_NAME2);
```

```
return 0;
```

```
}
```

```
void Test()
```

```
{
```

```
printf("%s/n",CONST_NAME2);
```

```
}
```

提示如下的编译信息：

Hello.c(15) : error C2065: 'CONST\_NAME1' : undeclared identifier

表示当前文件的名称被认为是Hello.c， #line 10 "Hello.c"所在的行被认为是第10行，因此提示第15行出错。

### 形式2

语法格式如下：

### # line constant

其作用在于编译的时候，准确输出出错代码所在的位置（行号），而在源程序中并不出现行号，从而方便程序员准确定位。

## 4：运算符#和##

在ANSI C中为预编译指令定义了两个运算符——#和##。

# 的作用是实现文本替换，例如

```
#define HI(x) printf("Hi,"#x"/n");
```

```
void main()
```

```
{
```

```
HI(John);
```

```
}
```

程序的运行结果

Hi,John

在预编译处理的时候, "#x"的作用是将x替换为所代表的字符序列。在本程序中x为John，所以构建新串“Hi,John”。

##的作用是串连接。

例如

```
#define CONNECT(x,y) x##y
```

```
void main()
```

```
{
```

```
int a1,a2,a3;
```

```
CONNECT(a,1)=0;
```

```
CONNECT(a,2)=12;
```

```
a3=4;
```

```
printf("a1=%d/ta2=%d/ta3=%d",a1,a2,a3);
```

}

程序的运行结果为

a1=0 a2=12 a3=4

在编译之前， CONNECT(a,1)被翻译为a1， CONNECT(a,2)被翻译为a2。

=====

**#define ,#undef** 分别用来定义常量、宏和取消常量、宏的定义。

**#include** 用来包含文件：

**#include <math.h>**与**#include “math.h”**的区别在于遇到**#include <math.h>**命令时系统从缺省的头文件目录中查找文件**math.h**文件；遇到**#include “math.h”**时系统首先从当前的目录中搜索，如果没有找到再在缺省的头文件目录中查找文件**math.h**文件。因此包含系统提供的库函数使用**#include <math.h>**方式搜索速度比较快；如果包含用户自定义的**.h**文件使用**#include “math.h”**方式,，搜索速度比较快。

提示 在使用**#include**指令的时候，对系统文件，使用**#include <math.h>**形式；对用户自定义文件，则使用**#include “math.h”**形式。

=====

条件编译系列：

条件编译

**#ifdef ... #else ...#endif**  
**#if defined... #else ...#endif**  
**#ifndef ... #else ...#endif**  
**#if !defined ... #else ...#endif**  
**#ifdef ...#elif ... #elif ...#else ... #endif**

=====

预编译系列：

预编译头文件说明

所谓头文件预编译，就是把一个工程(Project)中使用的一些MFC标准头文件(如Windows.H、 Afxwin.H)预先编译，以后该工程编译时，不再编译这部分头文件，仅仅使用预编译的结果。这样可以加快编译速度，节省时间。预编译头文件通过编译stdafx.cpp生成，以工程名命名，由于预编译的头文件的后缀是“pch”，所以编译结果文件是projectname.pch。

编译器通过一个头文件stdafx.h来使用预编译头文件。stdafx.h这个头文件名是可以在project的编译设置里指定的。编译器认为，所有在指令**#include "stdafx.h"**前的代码都是预编译的，它跳过**#include "stdafx. h"**指令，使用

projectname.pch编译这条指令之后的所有代码。

因此，所有的CPP实现文件第一条语句都是：**#include "stdafx.h"**。

另外，每一个实现文件**CPP**都包含了如下语句：

```
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

这是表示，如果生成调试版本，要指示当前文件的名称。**\_\_FILE\_\_**是一个宏，在编译器编译过程中给它赋值为当前正在编译的文件名称。

**VC**默认情况下使用预编译头(/Yu)，不明白的在加入新.h文件后编译时总出现**fatal errorC1010: 在查找预编译头指令时遇到意外的文件结尾的错误**。解决方法是在**include**头文件的地方加上**#include"stdafx.h"**,或者打项目属性，找到“**C/C++**”文件夹，单击“预编译头”属性页。修改“创建/使用预编译头”属性为“不使用预编译头”。

### VC的预编译功能

这里介绍**VC6**的预编译功能的使用，由于预编译详细使用比较的复杂，这里只介绍几个最重要的预编译指令: /Yu, /Yc,/Yx,/Fp。其它的详细资料可以参考： [MSDN -> Visual Studio 6.0 Document -> Visual C++ 6.0 Document -> VC++ Programmer Guider ->Compiler and Linker -> Details -> Creating Precompiled Header files](#)

预编译头的概念：

所谓的预编译头就是把一个工程中的那一部分代码，预先编译好放在一个文件里（通常是以.pch为扩展名的），这个文件就称为预编译头文件这些预先编译好的代码可以是任何的**C/C++**代码，甚至是**inline**的函数，但是必须是稳定的，在工程开发的过程中不会被经常改变。如果这些代码被修改，则需要重新编译生成预编译头文件。注意生成预编译头文件是很耗时间的。同时你得注意预编译头文件通常很大，通常有**6-7M**大。注意及时清理那些没有用的预编译头文件。

也许你会问：现在的编译器都有**Time stamp**的功能，编译器在编译整个工程的时候，它只会编译那些经过修改的文件，而不会去编译那些从上次编译过，到现在没有被修改过的文件。那么为什么还要预编译头文件呢？答案在这里，我们知道编译器是以文件为单位编译的，一个文件经过修改后，会重新编译整个文件，当然在这个文件里包含的所有头文件中的东西（.eg **Macro, Preprocessor** ）都要重新处理一遍。 **VC**的预编译头文件保存的正是这部分信息。以避免每次都要重新处理这些头文件。

根据上文介绍，预编译头文件的作用当然就是提高便宜速度了，有了它你没有必要每次都编译那些不需要经常改变的代码。编译性能当然就提高了。

要使用预编译头，我们必须指定一个头文件，这个头文件包含我们不会经常改变的代码和其他的头文件，然后用这个头文件来生成一个预编译头文件（.pch文件）想必大家都知道 **StdAfx.h**这个文件。很多人都认为这是**VC**提供的一个“系统级别”的，编译器带的一个头文件。其实不是的，这个文件可以是任何名字的。我们来考察一个典型的由**AppWizard**生成的**MFC Dialog Based** 程序的预编译头文件。（因为**AppWizard**会为我们指定好如何使用预编译头文件，默认的是**StdAfx.h**，这是**VC**起的名字）。我们会发现这个头文件里包含了以下的头文件：

```
#include <afxwin.h> // MFC core and standard components
#include <afxext.h> // MFC extensions
```

#include <afxdisp.h> // MFC Automation classes

#include <afxdtctl.h> // MFC support for Internet Explorer 4 Common Controls

#include <afxcmn.h>

这些正是使用MFC的必须包含的头文件，当然我们不太可能在我们的工程中修改这些头文件的，所以说他们是稳定的。

那么我们如何指定它来生成预编译头文件。我们知道一个头文件是不能编译的。所以我们还需要一个cpp文件来生成.pch 文件。这个文件默认的就是StdAfx.cpp。在这个文件里只有一句代码就是：**#include“Stdafx.h”**。原因是理所当然的，我们仅仅是要它能够编译而已———也就是说，要的只是它的.cpp的扩展名。我们可以用/Yc编译开关来指定StdAfx.cpp来生成一个.pch文件，通过/Fp编译开关来指定生成的pch文件的名字。打开project ->Setting->C/C++ 对话框。把Category指向Precompiled Header。在左边的树形视图里选择整个工程，Project Options(右下角的那个白的地方)可以看到 /Fp “debug/PCH.pch”，这就是指定生成的.pch文件的名字，默认的通常是 <工程名>.pch。然后，在左边的树形视图里选择StdAfx.cpp，这时原来的Project Option变成了 Source File Option（原来是工程，现在是一个文件，当然变了）。在这里我们可以看到 /Yc开关，/Yc的作用就是指定这个文件来创建一个Pch文件。/Yc后面的文件名是那个包含了稳定代码的头文件，一个工程里只能有一个文件的可以有YC开关。VC就根据这个选项把 StdAfx.cpp编译成一个Obj文件和一个PCH文件。

这样，我们就设置好了预编译头文件。也就是说，我们可以使用预编译头功能了。以下是注意事项：

- 1) 如果使用了/Yu，就是说使用了预编译，我们在每个.cpp文件的最开头，包含你指定产生pch文件的.h文件（默认是stdafx.h）不然就会有问题。如果你没有包含这个文件，就告诉你Unexpected file end.
- 2) 如果你把pch文件不小心丢了，根据以上分析，你只要让编译器生成一个pch文件就可以了。也就是说把stdafx.cpp（即指定/Yc的那个cpp文件）重新编译一遍就可以了。

顶 踩  
0 0

上一篇 C中的预编译宏定义

下一篇 全面了解setjmp与longjmp(C语言异常处理机制)

主题推荐

c语言

class

猜你在找

VC++游戏开发基础系列从入门到精通

《C语言/C++学习指南》语法篇（从入门到精通）

《C语言/C++学习指南》Linux开发篇

转C语言宏定义详解 - CC++

C语言中的预编译宏定义

C语言中的预编译宏定义

C++语言基础C语言连载四-----数组字符串函数递归预编译宏定义Python自动化开发基础 装饰器-异常处理-面向对象编程 c中有用的几个宏定义




查看评论

暂无评论

发表评论

用 户 名：zyp2524153

评论内容：



提交

\* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

- 全部主题HadoopAWS移动游戏JavaAndroidiOSSwift智能硬件DockerOpenStackVPNSparkERPIE10EclipseCRMJavaScript数据库UbuntuNFCWAPjQueryBIHTML5SpringApache.NETAPIHTMLSDKIISFedoraXMLLBSUnitySplashtopUMLcomponentsWindows MobileRailsQEMUKDECassandraCloudStackFTCcoremailOPhoneCouchBase云计算iOS6RackspaceWeb AppSpringSideMaemoCompuware大数据aptechPerlTornadoRubyHibernateThinkPHPHBasePureSolrAngularCloud FoundryRedisScalaDjangoBootstrap

