

九天雁翎的博客 做专业的程序, 做更专业的产品.

目录视图 摘要视图 RSS 订阅

个人资料



九天雁翎

访问: 2341725次
积分: 27268
等级: 
排名: 第137名

原创: 428篇 转载: 22篇
译文: 8篇 评论: 2818条

公告

此博客将为九天雁翎的博客(原程序即人生)的编程相关内容的镜像站. 扫描二维码, 关注作者:



我的链接

九天雁翎的博客(原程序即人生)
我的新浪微博
我的豆瓣
我的GitHub

链接

午后绿茶
酷壳
可能吧 (RSS)
clayman's blog (RSS)
实时渲染资源
史上最强游戏开发网站

文章搜索

CMake中的字符串及文件操作

标签: string output compiler regex file mercurial

2010-01-15 23:02 10568人阅读 评论(1) 收藏 举报

版权声明：本作品由九天雁翎创作，采用知识共享署名-非商业性使用 4.0 国际许可协议进行许可。
<http://www.jtianling.com>

目录(?) [\[+\]](#)

write by 九天雁翎(JTianLing) -- blog.csdn.net/vagrxie

[讨论新闻组及文件](#)

以前写过一篇关于CMake的基本用法的文章([现代软件构建系统的使用 cmake简介...](#)),在最近的使用中,发现一些更复杂情况的使用还有待学习,并且相关的资料也比较少,真正可以依赖的也就(CMake 实践)一文.所以,最近准备将一些非常重要,但是网上资料较少的用法--字符串,文件操作,好好整理一下.目标是,实现一个目录下的多个工程的一次性工程文件生成,而不是为每个工程都建立一个CMake文件,这个需求在我下载一些图书的配套源码时经常碰到,可能大部分的图书作者都习惯于仅提供原始源代码文件而不提供工程文件.

特别声明的是,本文并不是一个完备3的CMake教程(以前写的简介也不是),需要相关入门教程的应该去寻找(CMake 实践),那才是你想要的.本文仅仅是作为一些缺失资料的收集,也包括我自己对CMake字符串,文件处理的探索.(真是纯个人探索,网上即使是英文的相关资料也不多,可能毕竟CMake还是不怎么流行吧),仅适合需要的人搜索到这里查看相关信息.

本文内容:

实际上,本文研究的仅仅包括CMake的以下两部分,已经一些if,else中对字符串的判断.在本文中,由于目的不同,主要将会把CMake作为一种普通的编程语言来对待,(其实它本来就是)甚至很多地方根本不建工程,请注意.

string: String operations.

```
string(REGEX MATCH (regular_expression)
(output variable) (input) [(input)...])
string(REGEX MATCHALL (regular_expression)
(output variable) (input) [(input)...])
string(REGEX REPLACE (regular_expression)
(replace_expression) (output variable)
(input) [(input)...])
string(REPLACE (match_string)
(replace_string) (output variable)
(input) [(input)...])
string(COMPARE EQUAL (string1) (string2) (output variable))
string(COMPARE NOTEQUAL (string1) (string2) (output variable))
string(COMPARE LESS (string1) (string2) (output variable))
string(COMPARE GREATER (string1) (string2) (output variable))
string(ASCII (number) [(number) ...] (output variable))
string(CONFIGURE (string1) (output variable)
[@ONLY] [ESCAPE_QUOTES])
string(TOUPPER (string1) (output variable))
```

文章分类

- 【C++】 (126)
- 【Ruby】 (3)
- 【Python】 (19)
- 【JavaScript】 (1)
- 【C#】 (2)
- 【lua】 (14)
- 【Haxe】 (1)
- 【iPhone】 (9)
- 【Android】 (5)
- 【Fun】 (9)
- 【Linux】 (12)
- 【Windows】 (6)
- 【QT】 (11)
- 【随笔】 (69)
- 【产品设计】 (1)
- 【图形技术】 (36)
- 【我自己写的程序】 (27)
- 【汇编】 (13)
- 【游戏】 (33)
- 【书籍小评】 (5)
- 【算法】 (22)
- 【网络】 (23)
- 【翻译】 (13)
- 【转载】 (21)

文章存档

- 2013年08月 (1)
- 2013年07月 (1)
- 2013年06月 (1)
- 2013年05月 (1)
- 2013年04月 (2)

展开

阅读排行

- 小小游戏程序员工作两年 (61121)
- PyQt的学习（1）入门 (60026)
- Python与C之间的相互调 (57864)
- 网络协议编写的三层境界 (40647)
- Spine的使用(With Coco: (35219)
- 异常处理与MiniDump详 (32884)
- Google为啥没有Java的s (31558)
- Bullet物理引擎不完全指 (27817)
- SDL 简单入门学习 (26478)
- 新浪微博与腾讯微博的开 (24883)

```
string(TOLOWER (string1) (output variable))

string(LENGTH (string) (output variable))

string(SUBSTRING (string) (begin) (length) (output variable))

string(STRIP (string) (output variable))

string(RANDOM [LENGTH (length)] [ALPHABET (alphabet)]

(output variable))

file: File manipulation command.

file(WRITE filename "message to write"... )

file(APPEND filename "message to write"... )

file(READ filename variable [LIMIT numBytes] [OFFSET offset] [HEX])

file(STRINGS filename variable [LIMIT_COUNT num]

[LIMIT_INPUT numBytes] [LIMIT_OUTPUT numBytes]

[LENGTH_MINIMUM numBytes] [LENGTH_MAXIMUM numBytes]

[NEWLINE_CONSUME] [REGEX regex]

[NO_HEX_CONVERSION])

file(GLOB variable [RELATIVE path] [globbing expressions]...)

file(GLOB_RECURSE variable [RELATIVE path]

[FOLLOW_SYMLINKS] [globbing expressions]...)

file(REMOVE [file1 ...])

file(REMOVE_RECURSE [file1 ...])

file(MAKE_DIRECTORY [directory1 directory2 ...])

file(RELATIVE_PATH variable directory file)

file(TO_CMAKE_PATH path result)

file(TO_NATIVE_PATH path result)

file(DOWNLOAD url file [TIMEOUT timeout] [STATUS status] [LOG log])
```

以上列表来自于CMake最新的文档:<http://www.cmake.org/cmake/help/cmake2.6docs.html>

实验方式:

完全依靠CMake的GUI工具,通过configure的输出来获取实验结果.最终的结果就是为我的博客所有源代码重建工程,以后在博客源代码中仅保留CMake的配置文件,不再保留工程,保留工程很有局限性,比如Linux下没有makefile是很有问题的,个人建工程的时候用的是VS2008,这样VS2005的使用者也会很郁闷(虽然改个版本号就可以用),再加上以前刚开始学习OpenGL的时候非常短视,为了方便自己,将GLUT等库都装在系统目录下,这样的确是方便了自己,但是任何人clone的源代码都无法顺利运行,必须首先装一套配套的库才行,这可不是太好的工程管理方法,这次顺面解决.原来的源代码目录完全按照日期分目录,每天又分很多示例,很有典型意义.

比如说([DirectX 9.0 3D游戏开发编程基础](#))--([d3d龙书](#))(英文名:(Introduction to 3D Game Programming with DirectX 9.0))的附带源码仅有源代码文件,分散在N个目录(每章一个目录+每章每个示例一个目录),也与我的博客中源代码的分配很类似,事实上,很多书籍配套的源代码都有这样的问题,通过这样的方法可以一次解决.没有CMake的时候,想OpenGL红宝书,D3d龙书,3D游戏编程大师这样的书的源代码你得分别建几十个工程才能很好的在Windows下使用,这是非常让人郁闷的事情,你郁闷吗?我想,我可以通过这中方法搞定这些源代码,弄一个配套的project放在google code上,方便大家;),辛苦我一人,方便千万家的工作,那是责无旁贷的完成啊:)

本文顺面用于介绍CMake的强大及其推广,按照开源的理念,即使我没有真的参与某个开源产品的研发,我实际的使用了它,这是第一层次的支持,使用后感觉不错,向大家推广,这是第二层次的支持,我决定将我对CMake的支持上升一个层次(以前写简介的时候就上升过了)

实验过程

我的HelloWord:

获取某个源代码目录下的所有源代码,而不需要单个的列举,这是个很大的进步,在([现代软件构建系统的使用 cmake简介...](#))中介绍过,这是从

```
set(SRC_LIST test.cpp)
```

到

```
aux_source_directory(. SRC_LIST)
```

的进步.通过aux_source_directory来实现.

现在的问题是,单个工程这个函数很好用了,对于多个工程多级目录的情况呢?答案是file()一族的函数.

```
cmake_minimum_required(VERSION 2.8)
FILE(GLOB_RECURSE SRC_LIST "*.cpp")
FOREACH(src ${SRC_LIST})
MESSAGE( ${src} )
ENDFOREACH()
```

上面的代码,放在我的博客的根目录下,会找到所有的.cpp文件并输出

关键在于两个地方FILE(GLOB_RECURSE)

及FOREACH的使用

FILE(GLOB_RECURSE用于循环递归遍历目录并且使用GLOB匹配,GLOB是我们熟悉的文件匹配语法,(类似正则的简化版)

用于表示任意长度的任意文字,?用于匹配任意的一个字符.这里.cpp用于匹配所有的.cpp后缀的文件.

同时还有FILE(GLOB....用于非递归式的寻找,(即只寻找当前目录)

FOREACH就是我在使用C++时一直梦寐以求的循环语法,这里很明白,就是遍历\${SRC_LIST}的值,然后用MESSAGE输出,所以有了以下的输出.

在CMake的GUI中,确认正确的CMakeLists.txt的位置及工程准备在的位置,Configure后,输出一下信息:

Check for working C compiler: cl

Check for working C compiler: cl -- works

Detecting C compiler ABI info

Detecting C compiler ABI info - done

Check for working CXX compiler: cl

Check for working CXX compiler: cl -- works

Detecting CXX compiler ABI info

Detecting CXX compiler ABI info - done

F:/MySrc/blog/2009-10-12/SimpleRectangle/SimpleRectangle.cpp

F:/MySrc/blog/2009-10-14/GLDrawCircle/GLDrawCircle.cpp

F:/MySrc/blog/2009-10-14/GLDrawFourCircle/GLDrawFourCircle.cpp

F:/MySrc/blog/2009-10-14/GLFourCircleAnimation/GLFourCircleAnimation.cpp

F:/MySrc/blog/2009-10-14/GLShadeAnimation/GLShadeAnimation.cpp

F:/MySrc/blog/2009-10-14/WinDrawCircle/WinDrawCircle.cpp

F:/MySrc/blog/2009-10-18/gllInterleavedArrays/gllInterleavedArrays.cpp

F:/MySrc/blog/2009-10-18/glRectWithArrayDraw/glRectWithArrayDraw.cpp

```
F:/MySrc/blog/2009-10-18/glRectWithDrawElements/glRectWithDrawElements.cpp
F:/MySrc/blog/2009-10-18/glRectWithMultiDrawElements/glRectWithMultiDrawElements.cpp
F:/MySrc/blog/2009-10-18/glVertexArray/glVertexArray.cpp
F:/MySrc/blog/2009-10-18/glVertexArrayWithColor/glVertexArrayWithColor.cpp
F:/MySrc/blog/2009-10-20/OpenGL/debug/moc_opengl.cpp
F:/MySrc/blog/2009-10-20/OpenGL/main.cpp
F:/MySrc/blog/2009-10-20/OpenGL/opengl.cpp
F:/MySrc/blog/2009-10-21/gl3DCoordinate/gl3DCoordinate.cpp
F:/MySrc/blog/2009-10-21/glSmoothColorPyramid/glSmoothColorPyramid.cpp
F:/MySrc/blog/2009-10-21/glWirePyramid/glWirePyramid.cpp
F:/MySrc/blog/2009-10-25/glCullFace/glCullFace.cpp
F:/MySrc/blog/2009-10-25/glPolygonFace/glPolygonFace.cpp
F:/MySrc/blog/2009-10-25/glViewingTrans/glViewingTrans.cpp
F:/MySrc/blog/2009-10-26/glComposModelTrans/glComposModelTrans.cpp
F:/MySrc/blog/2009-10-26/glComposModelTrans2/glComposModelTrans2.cpp
F:/MySrc/blog/2009-10-26/glScaleSample/glScaleSample.cpp
F:/MySrc/blog/2009-10-28/glOrthoSample/glOrthoSample.cpp
F:/MySrc/blog/2009-10-28/glPerspective/glPerspective.cpp
F:/MySrc/blog/2009-10-29/JTFourTangram/JTTangram.cpp
F:/MySrc/blog/2009-11-11/glHalfTrans/glHalfTrans.cpp
F:/MySrc/blog/2009-11-12/glDepthTest/glDepthTest.cpp
F:/MySrc/blog/2009-11-12/glFogSample/glFogSample.cpp
F:/MySrc/blog/2009-11-9/glSimpleColorPyramid/glSimpleColorPyramid.cpp
F:/MySrc/blog/2009-11-9/LightSimple/LightSimple.cpp
F:/MySrc/blog/2009-9-27/Win32OpenGLTemplate/Win32OpenGLTemplate.cpp
F:/MySrc/blog/2009-9-28/RotateRect/RotateRect.cpp
F:/MySrc/blog/2009-9-29/TestOpenGL/mvarray.cpp
F:/MySrc/blog/JTTangram/JTTangram.cpp
```

Configuring done

获取了SRC_LIST后,就可以开始处理了

```
FOREACH(src ${SRC_LIST})
    IF(NOT (src MATCHES ".*2009-10-20.*"))
        MESSAGE( ${src} Finded)
        STRING( REGEX REPLACE ".*/(.*)/.cpp$" "/1" prjName ${src} )
        MESSAGE( ${prjName} Added)
        ADD_EXECUTABLE( ${prjName} ${src} )
```

```

IF (NOT (prjName STREQUAL mvarray) )
    SET_TARGET_PROPERTIES(${prjName} PROPERTIES WIN32_EXECUTABLE "true")
ENDIF(NOT (prjName STREQUAL mvarray) )

ENDIF(NOT (src MATCHES ".*2009-10-20.*"))
ENDFOREACH(src)

```

一下是关键点的解释:

通过STRING(REGEX REPLACE命令来获取到所有的.cpp后缀的文件,因为我的工程中所有的工程都仅包含一个cpp,所以可以这样做

```

string(REGEX REPLACE (regular_expression)
(replace_expression) (output variable)

```

的语法不算太好用,学会以后,就主要靠你的正则表达式理解能力了,几乎无所不能.CMake的语法中用()来表示正则表达式的匹配捕获,

用"1"的方式在替换中表示捕获到的字符串(会正则的理解起来应该不难).但是,特别需要注意的是,CMake中需要用/1的形式来传递参数,就像你在C++中需要做的那样.

```

ADD_EXECUTABLE( ${prjName} ${src} )

```

以每个cpp的文件名来作为工程名,每个源文件作为此工程需要的文件.

```

SET_TARGET_PROPERTIES(${prjName} PROPERTIES WIN32_EXECUTABLE "true")

```

用于指定我的这些工程都是Win32的工程(调用WinMain),默认时时命令行的工程.

上面这么几句就完成了为我10多个工程的一次工程文件生成

此代码中有几个特别点:

2009-10-20那个工程是一个Linux的工程,不要生成VS下win32的工程,我通过

```

IF (NOT (src MATCHES ".*2009-10-20.*"))

```

的形式过滤掉了

```

IF (NOT (prjName STREQUAL mvarray) )

```

语句用于过滤mvarray所在的工程,因为它是一个命令行的工程(使用Glut管理窗口)

以下是完整的文件(CMake的高亮很漂亮吧:)见我以前fachc2c),完成文件也可在我博客的示例代码中获取.

```

PROJECT( JTianLingBlog )
CMAKE_MINIMUM_REQUIRED(VERSION 2.8)
INCLUDE_DIRECTORIES(..dependes/windows/include/)
LINK_DIRECTORIES(..dependes/windows/lib/)
FILE(GLOB_RECURSE SRC_LIST "*.cpp")
FOREACH(src ${SRC_LIST})
    IF (NOT (src MATCHES ".*2009-10-20.*"))
        MESSAGE( ${src}  Finded)
        STRING( REGEX REPLACE ".*(/.*)/.cpp$" "/1" prjName ${src} )

```

进一步的应用

原来源代码的目录结构前面已经讲过了,直接先看全部CMake文件的代码:

已经比较复杂了,基本上,看到这里,你会认可其实CMake就是一个专门用于管理工程的脚本语言- -!

<http://blog.csdn.net/vagrxi/article/details/5196186>

用`FILE(GLOB_RECURSE srcList "*.cpp")` 遍历所有.cpp文件(带完整路径)

用`STRING(REGEX REPLACE "(.*\\.)/.*\\.cpp$" "/" dirName ${src})`获取到目录名

用`LIST(APPEND dirNameList ${dirName})`

保存所有的目录名,此时因为可能每个目录中有多个源代码导致重复,然后通过

`LIST(REMOVE_DUPLICATES dirNameList)`

将重复去掉

再通过

`STRING(REGEX REPLACE ".*/(.*)" "/" prjName ${dir})`

获取目录最后的名字作为每个工程的名字

`FILE(GLOB prjSrcList "${dir}/*.cpp" "${dir}/*.h")`

再反过来,获取每个目录下的源代码(此时已经获得了每个工程及其对应的源代码了)

剩下的就简单了

`STRING(REPLACE " " "" prj ${prjName})`

是唯一有意思的,去掉所有的空格后再建立工程(原来的目录中有空格,所以获取的工程名也有空格,并且导致建立的目录及一些文件会出现问题)

`TARGET_LINK_LIBRARIES(${prj} d3d9 d3dx9 winmm)`

链接上需要的库,OK,一下子生成几十个工程,还带`ALL_BUILD`选项,爽

为了方便大家,我将所有的([DirectX 9.0 3D游戏开发编程基础](#))(英文名:(Introduction to 3D Game Programming with DirectX 9.0)源代码托管在Google Code上了,并且建立好了VS2008的工程(不需要的用CMake换吧),大家各取所需吧.

浏览地址:

<http://code.google.com/p/jtianling/source/browse/?repo=introd3d9>

CheckOut地址:

hg clone <https://introd3d9.jtianling.googlecode.com/hg/> jtianling-introd3d9

下载地址:

[introd3d9.zip](#)

呵呵独家出品带工程的源代码:)

最后其实还有个问题,原来的源代码资源全部在各自的目录中,运行时的默认工作路径与工程一直,这样会全部找不到资源.但是我没有找到CMake怎么改变工作路径.

在网上找到一个CMake 的 [FAQ\(\)](#)是这样解释的:

Can CMake set the Debugging/Working Directory property in Visual Studio projects?

No. The value of this property is not stored in the project files. It is stored in extra files created by the IDE when a solution is loaded (VS .NET 2003 uses a hidden .suo file next to the .sln solution file). The format of these files is not known to CMake and cannot be generated. In some versions of VS the files are binary and not human readable.

事实上,我一直以为VS中将工作路径放在suo文件中而不是工程文件是MS干过的最愚蠢的事情之一,导致工作中的使用从来没有达到过从

源代码库中获取源代码及工程后,直接正常编译运行的情况,工作路径必须的额外设置,并且,要知道,是每个工程的工作路径!也因为这个原因,还因为suo是一个只有MS才懂的二进制文件,CMake无法自动改变此值,也就是说,虽然我已经通过CMake为你生成了几十个工程,但是凡事需要用到资源的地方,你都得手动改变工作路径,真无聊啊!!!!还好资源文件都不一样,我将其全部移到projects目录下,这样,只要你在projects目录下生成工程,默认的运行工作路径就是当前工作路径,省去了这样的操作,但是带来的资源及projects混合的情况,那就没有办法了.你们去骂MS吧,我已经尽力了.

事实上,虽然用CMake完成上述工作后,一次就建立了所有需要的工程,几十个几十个的生成,会感觉很爽,其实,花去的时间(主要用于学习CMake诡异特别的语法)要远超过自己一个一个手动生成工程.但是,实在无法忍受手动生成几十个工程的枯燥,习惯重复的工作只能使人越来越懒,没有任何收获.....

enjoy them all!

博客本身完整源代码获取说明

由于篇幅限制，本文一般仅贴出代码的主要关心的部分，代码带工程（或者makefile）完整版（如果有的话）都能用Mercurial在Google Code中下载。文章以博文发表的日期分目录存放，请直接使用Mercurial克隆下库：

<https://blog-sample-code.jtianling.googlecode.com/hg/>

Mercurial使用方法见《分布式的，新一代版本控制系统Mercurial的介绍及简要入门》

要是仅仅想浏览全部代码也可以直接到google code上去看，在下面的地址：

<http://code.google.com/p/jtianling/source/browse?repo=blog-sample-code>

原创文章作者保留版权 转载请注明原作者 并给出链接

write by 九天雁翎(JTianLing) -- blog.csdn.net/vagrxie

顶 0 踩 0

- 上一篇 可商业使用的免费软件推荐列表（ revision 2 ）
- 下一篇 C和Python程序员的JavaScript学习指南（译）

参考知识库



Git知识库


682 关注 | 331 收录

猜你在找

- | | |
|------------------|-------------------------|
| OpenGL ES2.0基础 | GitHub 优秀的 Android 开源项目 |
| iOS8-Swift开发教程 | C++工程实践经验 |
| 深入浅出Unity3D——第一篇 | 跟我一起写 Makefile |
| 三维游戏引擎开发-渲染 | 跟我一起写 Makefile |
| 韦东山嵌入式Linux第一期视频 | 跟我一起写 Makefile |

查看评论

1楼 [lgnt](#) 2010-08-15 20:49发表



STRING(REGEX REPLACE "(*/*).*/.cpp\$" "1" ; dirName \${src})中的“1”代表的是什么意思？

Re: [九天雁翎](#) 2010-08-16 11:27发表



学习正则表达式你才会知道。。。。。。表示前面匹配字符串中的第一个()中匹配的内容

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)