**Boost编译和安装说明**

**一、下载和安装**

从官方主页http://www.boost.org下载最新版，因为boost一部分类是需要编译成库才能使用的，所以需要准备好boost专用的编译辅助工具bjam。可直接使用boost安装包中附带的bjam源码来编译出bjam。

将boost安装包解压到E:\boost\_1\_53\_0，

运行E:\boost\_1\_53\_0\bootstrap.bat批处理，会复制E:\boost\_1\_53\_0\tools\build\v2\engine\

bin.ntx86\bjam.exe到目录E:\boost\_1\_53\_0\下，注意最后bjam.exe必须与boost-build.jam在同级目录。

**二、编译**

打开命令提示符（cmd.exe）窗口并执行bjam，使用--help查看命令帮助。bjam的命令行参数涉及具体库的选择，会影响编译后的硬盘空间。

|  |
| --- |
| 使用bjam命令如：bjam stage --toolset=msvc-10.0 --without-python  --without-wave --without-test --stagedir="E:\boost\_1\_53\_0\_vc10"  link=shared runtime-link=shared threading=multi debug release |

下面详细解释一下每个参数的含义：

1. stage/install：stage表示只生成库（dll和lib），install还会生成包含头文件的include目录。推荐使用stage，因为install生成的这个include目录实际就是boost安装包解压缩后的boost目录（E:\boost\_1\_53\_0\boost，只比include目录多几个非hpp的小文件），可以直接使用，而且不同的IDE都可以使用同一套头文件。
2. toolset：指定编译器，可选的如borland、gcc、msvc（VC6）、msvc-9.0（VS2008）、msvc-10.0（VS2010）等。
3. without/with：选择不编译/编译哪些库，默认是全部编译。例如，without-python意思是不需要编译python库。如果选择编译python的话，应该到python官方主页http://www.python.org下载安装。
4. stagedir/prefix：stage时使用stagedir，install时使用prefix，表示编译生成文件的路径。如果使用了install参数，那么还将生成头文件目录。推荐给不同的IDE指定不同目录
5. build-dir：编译生成的中间文件的路径。默认就在根目录（E:\boost\_1\_53\_0）下，目录名为bin.v2，等编译完成后可具体查看一下是否生成了lib、dll文件。如果不需要可直接删除bin.v2目录
6. link：生成动态链接库/静态链接库。动态链接库使用shared方式，静态链接库使用static方式。注意，static方式下生成的很多静态链接库大小都在几兆、几十兆。不推荐以static方式编译（without掉），巨型库黑名单：wave、graph、math、regex、test、program\_options、serialization、signals。
7. runtime-link：动态/静态链接C/C++运行时库，有shared和static两种方式，与link参数组合有4种方式。但通常一个工程用动态链接那么所有库都用动态链接，如果用静态链接那么所有库都用静态链接。所以只需要编译2种组合即可，即link=shared runtime-link=shared和link=static runtime-link=static。
8. threading：单/多线程编译。一般使用多线程程序，要指定multi方式；如果需要单线程环境，使用single方式。
9. debug/release：编译debug/release版本，一般两个都编译出来。

|  |
| --- |
| 编译总共用时20分钟左右。分别编译静态链接和动态链接版本后，整个E:\boost\_1\_53\_0目录（包括安装包解压缩文件和编译生成的库文件）只有250MB。一般保留E:\boost\_1\_53\_0\boost 头文件目录、libs目录，它提供了所有Boost类的使用范例参考；doc目录有完整的boost使用帮助。其他目录和文件可以删除，节约出约150MB空间。如果图省事，可以使用命令：bjam --toolset=msvc-10.0 --build-type=complete |

指定编译器以完全模式编译，编译所有库，带来的后果是：

1. 占用3G以上的硬盘空间
2. 占用若干小时的编译时间
3. 头文件和库文件都存放在C:\Boost目录下,比较混乱
4. 生成的很多文件可能永远也用不上

**三、配置**

include目录：E:\boost\_1\_53\_0\boost，约90MB

library目录：E:\boost\_1\_53\_0vc10，编译后的dll动态库约15MB，加上dll动态库调试符号pdb文件，约95MB

添加到IDE相应的路径下面即可。

**四、使用**

使用举例：

|  |
| --- |
| #include <boostthread.hpp> |

此时不用包含库文件，boost的auto-link机制将会自动包含对应的静态lib。即boost默认是以静态方式链接的，工程属性最好设为Multi-threaded (Debug)。如果想使用dll动态方式链接，需要预先定义宏：

|  |
| --- |
| #define BOOST\_ALL\_DYN\_LINK |

此时boost会默认包含对应的lib。如果不想使用boost提供的auto-link机制，可以预先定义宏：

|  |
| --- |
| #define BOOST\_ALL\_NO\_LIB |

然后使用以下方法链接：

|  |
| --- |
| #pragma comment(lib, "boost\_thread-vc100-mt-1\_53.lib") |

或

|  |
| --- |
| #pragma comment(lib, "boost\_thread-vc100-mt.lib") |

这两个lib其实是一样的，实在不明白boost编译时为什么每个库都要复制一份？另外还有一个比较有用的宏：

|  |
| --- |
| #define BOOST\_LIB\_DIAGNOSTIC |

它让VC在编译时的output窗口中输出具体链接了哪些boost库及链接顺序。

关于boost的auto-link机制，详情可查看boostconfigauto\_link.hpp里的代码。

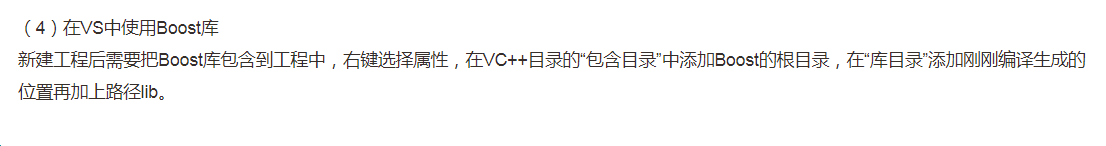
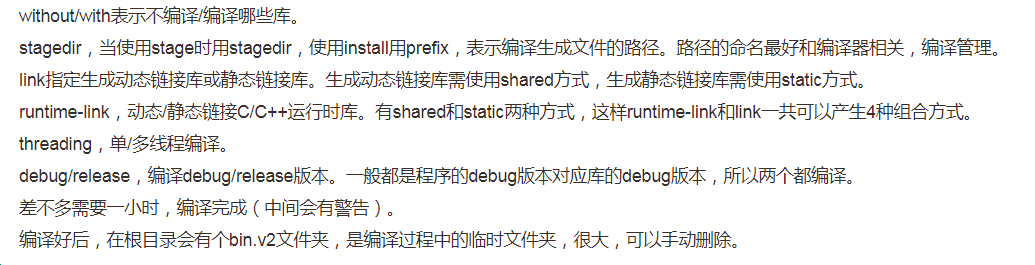
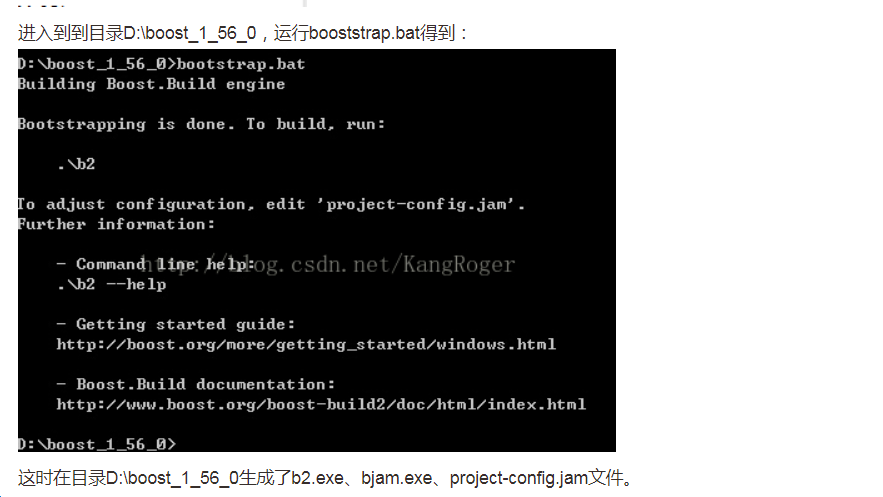
Boost库编译

参考网友：deep\_learninger

<http://blog.csdn.net/u014114990/article/details/51394973>







参考另一位网友

<http://blog.csdn.net/twtydgo/article/details/51672682>



