1.2 关于 STLport

本节将介绍 C++标准库的一个高效实现——STLport，可以用于配合 Boost 程序库工作。

1.2.1 什么是 STLport

STLport 是一个完全符合 C++98 标准（及 2003 年修订）的一个免费的 C++标准库实现。它是由俄罗斯人 Boris Fomitchev 于 1997 年发起的开源项目，目的是基于著名的 SGISTL 开发一个可移植到各种平台上使用的高效的 C++标准库。

STLport 具有很多其他 STL 实现所没有的优点。首先是高度的可移植性，可以配合市面上几乎所有的操作系统和编译器使用，使开发的程序能够在不同的编译平台上获得一致的标准库实现。其次是性能表现优秀，其原始版本 SGISTL 就以高效而闻名，STLport 在移植时也特别注重性能与效率，而且 100%完全符合 C++98 标准规范。第三个优点是在标准之外增加了若干有用的扩展，如 rope（增强的字符串类）、slist（单链表数据结构）、hash_map（散列映射容器），以及支持线程安全。

STLport 以其优异的品质自发布以来获得了极大的成功，以至于 Boost 专门为 STLport 提供了编译选项和设置。[①]

Windows 平台开发主流工具是 MSVC，其自带的 STL 向来名声不佳，虽然随着 VC 的版本升级而逐渐改善，但质量仍非一流水准。作者曾经在工作用机上运行过简单的自测，结果是 VC8 自带的 STL（Dinkumware v405）较 STLport5.21 慢大约一倍；而 VC9 自带的 STL（Dinkumware v503）速度虽然有较大改善，基本与 STLport5.21 速度相当，但仍有大约 10%以上的差距。综合各个方面来看，STLport 都较 VC 自带的 Dinkumware STL 实现好很多。

基于以上原因，本书在 Windows 平台上使用 STLport 搭配 Boost，而没有使用 VC8 自带的 Dinkumware STL。

1.2.2 安装 STLport

STLport 目前的最新版本是 2008 年 12 月 10 日发布的 5.21 版。

---

① 遗憾的是 STLport 自从 2008 年的 5.21 版之后就停止开发了，未能支持 C++11，但 Boost 库仍然保留了对它的支持。

从 STLport 网站（http://stlport.sourceforge.net/）下载压缩包 STLport-5.2.1.tar.bz2，解压缩到硬盘任意位置即可，本书使用的路径是 D:\STLport。

1.2.3 编译 STLport

本书使用默认选项编译生成 STLport 的多线程库，具体步骤如下：

1．从“开始”菜单运行 VS2005 工具的命令行提示符“Visual Studio 2005 Command Prompt”。

2．执行命令“cd D:\STLport”，进入 D:\STLport 目录。

3．执行命令“configure msvc8”（如果使用 VC6 则执行 configure msvc6）配置编译环境。

4．执行命令“cd D:\STLport\build\lib”。

5．执行命令“nmake -f msvc.mak clean install”。

STLport 编译时间稍长，大约需要数分钟。编译完成后会自动将编译出的*.dll 和*.lib 复制到 STLport\lib 和 STLport\bin 目录下，之后可将 STLport\build\lib\obj 目录删除以节约硬盘空间。

1.2.4 使用 STLport

STLport 早期（5.0 版以前）支持对本地 IO 流的 wrapper 模式，可以不需要编译，简单地定义一个宏即可使用。但自从 5.0 版后 STLport 取消了这个模式，用户只能选择使用流或者不使用流（如有的平台不需要流输出的情形），这也就意味着 STLport 必须要编译后才能使用。

在 Debug 模式下使用 STLport，需要定义宏“__STL_DEBUG”。

在 Debug 模式下与 Boost 配合使用 STLport，需要定义宏“_STLP_DEBUG”。

与 MFC 配合使用 STLport，需要定义宏“_STLP_USE_MFC”。

更多有关 STLport 的使用信息请参阅 STLport 的说明文档。

1.3 开发环境简介

C++是一个大型语言，十分复杂。虽然 C++98 标准已经面世十余年，C++11 标准也已经出台一年多，但仍然没有一个编译器敢宣称自己能够 100%支持 C++的全部特性。

由于 Boost 大量使用了 C++高级特性（如模板偏特化、ADL），因此不是所有的编译器都能够很好地支持 Boost，并且每个组件对编译器的支持都不尽相同。虽然 Boost 已经针对平台和编

译器的兼容性做了大量的工作，但仍有可能出现意外情况（某些过“老”的编译器已经不再被支持）。

阅读本书和使用 Boost，读者需要一个能够较好地支持 C++标准（至少是 C++98 标准）的编译器和开发环境。

本书作者使用了如下四个开发环境（操作系统+开发工具+编译器+标准库），大部分代码均在这些环境中编译通过：

1) Mac OS X 10.8.2，Xcode4.4（Clang4.0），libc++；

2) Ubuntu 12.04 LTS（Linux 3.2.0），GCC4.6.3 ，libstdc++；

3) Windows7，MinGW（GCC4.7.1），libstdc++；

4) Windows7，Visual Studio 2005（VC8），STLport5.21。[①]

以上四个环境中除最后一个环境比较“老”以外，其余三个都可以较好地支持新的 C++11 标准。

对于使用其他开发环境的读者只能说抱歉了，作者不能保证书中代码能够百分之百正确运行。请参考 Boost 说明文档查看对您正在使用的平台或编译器的支持情况。

1.4 开发环境搭建

本节简略介绍在上述开发环境里如何编译、设置 Boost 程序库，供读者参考。

1.4.1 UNIX 开发环境

Boost1.51 的编译不再使用早期版本的 bjam，而是改用了新的 b2 工具（boost build v2），所以编译较以前版本有了很大的简化，速度也很快。

编译 Boost

UNIX 系统（OS X、Linux）下编译 Boost 很容易，最简单省事的编译方法是在 Boost 安装目录下直接执行命令：

```
./bootstrap.sh;./b2
```

[①] 作者并没有使用微软最新的 VS2010 或 VS2012，主要是考虑 VC8 是一个支持 C++98 而不支持 C++11 的编译器，特意用它来验证 Boost 的跨标准兼容性。

第一条命令 bootstrap.sh 是编译前的配置工作，第二条命令 b2 开始真正的编译。

得益于摩尔定律，现在 Boost 库的编译所需要的时间和空间都已经大大缩减了，在目前主流级别 CPU 上只需要半小时左右，而在以前则需要数个小时。

我们也可以完整编译 Boost，需要在 bootstrap.sh 之后执行如下命令：

```
./b2 --buildtype=complete stage
```

这样将开始对 Boost 的完整编译，生成所有调试版、发行版的静态库和动态库。

其中：

- buildtype 选项指定编译类型，如不指定则默认使用 release 模式；
- stage 选项指定 Boost 使用本地构建。如果使用 install 选项则编译后会把 Boost 安装到默认路径下（/usr/local）。

MakeFile 范例

UNIX 下最常用的构建工具是 make，所以下面给出本书在 Mac OS X 下使用的一个 makefile 代码，非常简单，仅作示意：

```
CC = clang                                      #Linux 应使用 gcc
XX = clang++                                    #Linux 应使用 g++
CFLAGS  = -Wall -D_REENTRANT -g -std=c++11      #Linux 应使用 gnu++0x
INCLUDE = -I/Users/chrono/boost/                #Boost 库安装在用户主目录
STDLIB  = -stdlib=libc++                        #Linux 无需此 FLAG

LIBS = -lpthread -lrt                           #posix 线程库和实时库
OBJS = main.o
TEST = test

%.o: %.c
  $(CC) $(CFLAGS) $(INCLUDE) -c $< -o $@
%.o:%.cpp
  $(XX) $(CFLAGS) $(STDLIB) $(INCLUDE) -c $< -o $@

all: $(TEST)

$(TEST): $(OBJS)
  $(XX) $(STDLIB) -o $(TEST) $(OBJS) $(LIBS)
  ./$(TEST)                                     #编译后直接运行程序

clean:
  rm -f *.o
```

1.4.2 Windows 开发环境

Windows 系统下使用 VC 编译 Boost 的方法与 UNIX 类似，但因为使用 STLport 所以要稍微麻烦一些，不过也不难。

编译前的配置

首先要在 Boost 安装目录下执行命令：

```
bootstrap
```

稍等片刻就会完成编译前的配置工作。

修改 Boost 配置，启用 STLport

修改 b2 的配置文件：\tools\build\v2 下的 user-config. jam，在第 74 行去掉前面的#注释，启用 STLport，并修改 STLport 的头文件路径和 lib 路径，例如改成 “using stlport : : d:\stlport\stlport : d:\stlport\lib”。

如果读者不打算采用 STLport 作为 C++标准库的替代，那么本步骤可以省略。

编译 Boost

完成如上准备工作（bootstrap 和修改配置），接下来就可以开始正式编译 Boost 库了，同 UNIX 一样可以直接执行命令：

```
b2
```

而完整编译 Boost 需要执行如下命令：

```
b2 --buildtype=complete stdlib=stlport stage
```

其中多了一个 stdlib 选项指定要搭配的标准库，如不使用 STLport 可不用该选项。

Visual Studio 环境设置

在编译完 STLport 和 Boost 后，还需要设置 VC 的环境选项，才能让 VC 识别 STLport 和 Boost 从而正常使用。

本书采用静态库链接、多线程、非 Unicode 的编译方式：

- 打开菜单 Tools->Options，在“Projects and Solutions”的“VC++ Directories”页，选择 Include files，加入 D:\STLport\stlport 和 D:\boost\，并调到最前面；选择 Library files，加入 D:\STLport\lib，并调到最前面。
- 打开菜单 Project->Properties，在“Configuration Properties”的“General”页，设置 Character

Set 为 Not Set。

- 在“C/C++”的“Code Generation”页，选择 Runtime Library 为多线程（release 版是/MT，Debug 版为/MTd）。
- 如果是 Debug 版工程，不要忘记在 Preprocessor 页中定义宏“_STLP_DEBUG”和“__STL_DEBUG”以使用 STLport。

1.4.3 高级议题

本节讨论关于 Boost 编译的一些高级议题。

Boost 库的命名规则

Boost 库的文件名遵循的规则十分清晰明了，基本形式如下：

```
libboost_filesystem-vc80-mt-sgdp-1_51.lib
```

前缀 ：统一为 lib，但在 Windows 下只有静态库有 lib 前缀；

库名称 ：以“boost_”开头的库名称，在这里是 boost_filesystem；

编译器标识 ：编译该库文件的编译器名称和版本，在这里是-vc80；

多线程标识 ：支持多线程使用-mt，没有表示不支持多线程；

ABI 标识 ：这个标识比较复杂，标识了 Boost 库的几个编译链接选项；

- s ：静态库标识
- gd：debug 版标识
- p ：使用 STLport 而不是编译器自带 STL 实现

版本号 ：Boost 库的版本号，小数点用下画线代替，在这里是 1_51；

扩展名 ：在 Windows 上是.lib，在 Linux 等类 UNIX 操作系统上是.a 或者.so。

对于 Clang、GCC 等编译器，在链接 Boost 程序库时可以有两种方式：一种是在编译命令行直接指明库文件全路径，另一种是用-L 指定库文件所在路径，再用-l 指定库文件名。

Boost 库在 VC 编译器下支持库自动链接技术（使用#pragma comment(lib, XXX)），只要把所有生成的 lib 拷贝到 VC 的搜索路径下，不需要你费心，编译器会自动根据编译选项找到合适的库链接成可执行文件。

部分编译 Boost

完整编译 Boost 费时费力，而且这些库并不可能在开发过程中全部用到，因此，b2 也允许用户自行选择要编译的库。

执行“b2 --show-libraries”，可查看所有必须编译才能使用的库。在完全编译命令的基础上，使用--with 或者--without 选项可打开或者关闭某个库的编译，如：

```
./b2 --with-date_time --buildtype=complete stage
```

将仅编译 date_time 库。

b2 还有其他很多选项，如指定安装路径、指定 debug 或 release 版等。对 b2 的进一步介绍已经超出了本书的范围，读者可使用--help 选项或者参考 Boost 文档以获得更多信息，本书不再叙述。

unity build

编译库的方法使用 Boost 费时费力，有时还要面临链接的烦恼。本书推荐使用作者自行实践的另一种方式：unity build——把 Boost 源代码嵌入到自己的工程中编译。

这种方法的原理类似 VC 的预编译技术，在工程中直接包含 Boost 库的 cpp 文件，不但可以省略库的编译步骤，而且源代码还获得了独立于编译器、操作系统和 Boost 库版本的好处（任何一个因素发生变化时都不需要换编译器重新编译库），能够增强程序的可移植性。

不过这种方法也有小小的代价，就是每个工程都要重新编译 Boost 库，不如前两种方式那样可以（暂时的）“一劳永逸”，但作者认为其平台无关的优点还是大于缺点。在其后的章节里会向读者示范这种嵌入工程编译的方式如何具体实现，这里暂举一个小例子：

```
/// sysprebuild.cpp 一个嵌入编译源代码文件
#define BOOST_SYSTEM_NO_LIB                          //禁用 Boost 的自动链接功能
#include <libs/system/src/error_code.cpp>            //包含嵌入编译源代码
```

以上代码实现了 boost.system 库的嵌入编译，读者只需要将该文件（sysprebuild.cpp）加入到工程（或者 makefile）中，无须使用 b2 预先编译库就能享用 Boost 带来的好处。即使将来 Boost 程序库更新版本，也只需要简单地重新编译工程即可使用新版的功能，而无需耗费大量时间去重新编译链接 Boost 库。