POCO C++库学习和分析 -- 文件系统

作者：[**arau\_sh**](http://my.csdn.net/arau_sh)

 既然作为一个框架性的库，自然会提供对于文件系统的操作。在Poco库中，封装了一些类去完成上述操作。这些类包括了：  
              1. Poco::Path

              2. Poco::File

              3. Poco::TemporaryFile

              4. Poco::DirectoryIterator

              5. Poco::Glob

              这些类在实现上并没有什么特殊的注意点，主要是不同操作系统API的调用。如果想学习API函数的话，确实是一个不错的例子。在这里将主要介绍这些类的接口和使用，主要以翻译Poco的使用文档为主。

## 1. Poco::Path

### 1.1 路径：

               1. 在不同操作系统中，指明文件和目录所在位置的标示符是不一样的。  
               2. 标示符的不一致，会造成代码在不同平台之间移植的困难。  
               3. Poco::Path类抽象了不同标识符之间的区别，使程序员可以把注意力集中在业务的开发上。  
               4. Poco::Path类支持Windows、Unix、OpenVMS操作系统。

### 1.2 Poco路径简介：

**Poco中的路径包括了**：  
               1. 一个可选的节点(node)名:  
                              a) 在Windows上，这是计算机在UNC([Universal Naming Convention](http://baike.baidu.com/view/244940.htm))路径中的名字  
                              b) 在OpenVMS中，这代表一个集群系统中的节点名  
                              c) 在Unix中，此名字未被使用。  
               2. 一个可选的设备(device)名:  
                              a) 在Windows上，这是一个驱动器盘符  
                              b) 在OpenVMS上，这是存储盘符的名字  
                              c) 在Unix，此名字未被使用。  
               3. 一个目录名的列表  
               4. 一个文件名(包括扩展名)和版本号(OpenVMS特有)  
  
               **Poco支持两种路径**:  
               1. 绝对路径  
               以根目录为起点的描述资源的目录  
               2. 相对目录  
               以某一个确定路径为起点的描述资源的目录(通常这是用户的当前目录)  
  
               相对目录可以被转换为绝对目录(反之，并不成立)。  
               在Poco中路径的指向可以是一个目录也可以是一个文件。当路径指向目录时，文件名为空。  
  
               下面是Poco中关于路径的一些例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. Path: C:\Windows\system32\cmd.exe
2. Style: Windows
3. Kind: absolute, to file
4. Node Name: –
5. Device Name: C
6. Directory List: Windows, system32
7. File Name: cmd.exe
8. File Version: –
10. Path: Poco\Foundation\
11. Style: Windows
12. Kind: relative, to directory
13. Node Name: –
14. Device Name: –
15. Directory List: Poco, Foundation
16. File Name: –
17. File Version: –
19. Path: \\www\site\index.html
20. Style: Windows
21. Kind: absolute, to file
22. Node Name: www
23. Device Name: –
24. Directory List: site
25. File Name: index.html
26. File Version: –
28. Path: /usr/local/include/Poco/Foundation.h
29. Style: Unix
30. Kind: absolute, to file
31. Node Name: –
32. Device Name: –
33. Directory List: usr, local, include, Poco
34. File Name: index.html
35. File Version: –
37. Path: ../bin/
38. Style: Unix
39. Kind: relative, to directory
40. Node Name: –
41. Device Name: –
42. Directory List: .., bin
43. File Name: –
44. File Version: –
46. Path: VMS001::DSK001:[POCO.INCLUDE.POCO]POCO.H;2
47. Style: OpenVMS
48. Kind: absolute, to file
49. Node Name: VMS001
50. Device Name: DSK001
51. Directory List: POCO, INCLUDE, POCO
52. File Name: POCO.H
53. File Version: 2

### 1.3 类说明

               1. Poco::Path类在Poco库中代表了路径。  
               2. Poco::Path类并不关心路径所指向的目标在文件系统中是否存在。这个工作由Poco::File类负责。  
               3. Poco::Path支持值语义(copy函数和赋值函数)，但不支持关系操作符。  
  
             **构建一个路径**  
               构建一个路径存在着两种方式：  
               1. 从0开始构建，分别构建node、device、directory、file  
               2. 通过一个包含着路径的字符串去解析  
  
               在构建时，可以指定路径的格式：  
                   a)PATH\_UNIX  
                   b)PATH\_WINDOWS  
                   c)PATH\_VMS  
                   d)PATH\_NATIVE (根据当前系统格式判断)  
                   e)PATH\_GUESS (让Poco库自行判断)  
  
  
               从0构造路径  
               1. 创建一个空路径，使用默认的构造函数(默认情况下路径格式为"相对目录")或者构造时使用一个bool参数去指定路径格式(true = absolute, false = relative)  
               2. 如果需要的话，使用下列赋值函数去设置节点和设备名  
                              void setNode(const std::string& node)  
                              void setDevice(const std::string& device)  
               3. 添加路径名  
                              void pushDirectory(const std::string& name)  
               4. 设置文件名  
                              void setFileName(const std::string& name)  
  
               下面是一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. #include "Poco/Path.h"
2. **int** main(**int** argc, **char**\*\* argv)
3. {
4. Poco::Path p(**true**); // path will be absolute
5. p.setNode("VMS001");
6. p.setDevice("DSK001");
7. p.pushDirectory("POCO");
8. p.pushDirectory("INCLUDE");
9. p.pushDirectory("POCO");
10. p.setFileName("POCO.H");
11. std::string s(p.toString(Poco::Path::PATH\_VMS));
12. // "VMS001::DSK001:[POCO.INCLUDE.POCO]POCO.H"
13. p.clear(); // start over with a clean state
14. p.pushDirectory("projects");
15. p.pushDirectory("poco");
16. s = p.toString(Poco::Path::PATH\_WINDOWS); // "projects\poco\"
17. s = p.toString(Poco::Path::PATH\_UNIX); // "projects/poco/"
18. s = p.toString(); // depends on your platform
19. **return** 0;
20. }

               从一个字符串中解析路径名  
               1. Poco支持从一个字符串中解析路径名  
                              Path(const std::string& path)  
                              Path(const std::string& path, Style style)  
               如果函数调用时，路径格式style不被指定，将使用当前系统路径格式。  
               2. 可以从另一个路径(指向目录名)和文件名，或者两个路径(第一个为绝对路径，第二个为相对路径)构造  
                              Path(const Path& parent, const std::string& fileName)  
                              Path(const Path& parent, const Path& relative)  
  
               路径也可以通过下列函数去构建  
                              Path& assign(const std::string& path)  
                              Path& parse(const std::string& path)  
                              Path& assign(const std::string& path, Style style)  
                              Path& parse(const std::string& path, Style style)  
  
               如果路径非法的话，会抛出Poco::PathSyntaxException异常。想要测试一个路径字符串是否合法，可以使用tryParse()函数：  
                              bool tryParse(const std::string& path)  
                              bool tryParse(const std::string& path, Style style)  
  
               下面是一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. #include "Poco/Path.h"
2. **using** Poco::Path;
3. **int** main(**int** argc, **char**\*\* argv)
4. {
5. //creating a path will work independent of the OS
6. Path p("C:\\Windows\\system32\\cmd.exe");
7. Path p("/bin/sh");
8. p = "projects\\poco";
9. p = "projects/poco";
10. p.parse("/usr/include/stdio.h", Path::PATH\_UNIX);
11. **bool** ok = p.tryParse("/usr/\*/stdio.h");
12. ok = p.tryParse("/usr/include/stdio.h", Path::PATH\_UNIX);
13. ok = p.tryParse("/usr/include/stdio.h", Path::PATH\_WINDOWS);
14. ok = p.tryParse("DSK$PROJ:[POCO]BUILD.COM", Path::PATH\_GUESS);
15. **return** 0;
16. }

               Poco::Path类提供了函数用于转换成为字符串：  
               std::string toString()  
               std::string toString(Style style)  
               当然也可以使用下列函数得到路径不同部分的字符串：  
              const std::string& getNode()  
              const std::string& getDevice()  
              const std::string& directory(int n) (also operator [])  
              const std::string& getFileName()  
               可以调用下列函数获取目录的深度:  
               int depth() const  
  
               通过下面的函数可以得到和设置文件的基本名和扩展名:  
                              std::string getBaseName() const  
                              void setBaseName(const std::string& baseName)  
                              std::string getExtension() const  
                              void setExtension(const std::string& extension)  
  
               下面是一个例子:

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. #include "Poco/Path.h"
2. **using** Poco::Path;
3. **int** main(**int** argc, **char**\*\* argv)
4. {
5. Path p("c:\\projects\\poco\\build\_vs80.cmd", Path::PATH\_WINDOWS);
6. std::string device(p.getDevice()); // "c"
7. **int** n = p.depth(); // 2
8. std::string dir1(p.directory(0)); // "projects"
9. std::string dir2(p[1]); // "poco"
10. std::string fileName(p[2]); // "build\_vs80.cmd"
11. fileName = p.getFileName();
12. std::string baseName(p.getBaseName()); // "build\_vs80"
13. std::string extension(p.getExtension()); // "cmd"
14. p.setBaseName("build\_vs71");
15. fileName = p.getFileName(); // "build\_vs71.cmd"
16. **return** 0;
17. }

 **路径操作:**  
               1. Path& makeDirectory()  
               确保路径的结尾是一个目录名。如果原路径有文件名存在的话，添加一个与文件名同名的目录，并清除文件名。  
               2. Path& makeFile()  
               确保路径的结尾是一个文件名。如果原路径是一个目录名，则把最后一个目录名变成文件名，并去除最后一个目录名。  
               3. Path& makeParent()  
                   Path parent() const  
               使路径指向它的父目录(如果存在文件名的话，清除文件名；否则的话则移除最后一个目录名)  
               4. Path& makeAbsolute()  
                   Path& makeAbsolute(const Path& base)  
                   Path absolute() const  
                   Path absolute(const Path& base)  
               转换相对路径为绝对路径  
               5. Path& append(const Path& path)  
               添加路径  
               6. Path& resolve(const Path& path)  
               如果新的路径为绝对路径，则代替现有的路径；否则则在原路径下追加  
  
**路径属性：**  
               1. bool isAbsolute() const  
               如果路径为绝对路径，返回true；否则为false  
               2. bool isRelative() const  
               如果路径为相对路径，返回true；否则为false  
               3. bool isDirectory() const  
               如果路径为目录，返回true；否则为false  
               4. bool isFile() const  
               如果路径为文件，返回true；否则为false  
  
  
               下面是一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. #include "Poco/Path.h"
2. **using** Poco::Path;
3. **int** main(**int** argc, **char**\*\* argv)
4. {
5. Path p("/usr/include/stdio.h", Path::PATH\_UNIX);
6. Path parent(p.parent());
7. std::string s(parent.toString(Path::PATH\_UNIX)); // "/usr/include/"
8. Path p1("stdlib.h");
9. Path p2("/opt/Poco/include/Poco.h", Path::PATH\_UNIX);
10. p.resolve(p1);
11. s = p.toString(Path::PATH\_UNIX); // "/usr/include/stdlib.h"
12. p.resolve(p2);
13. s = p.toString(Path::PATH\_UNIX); // "/opt/Poco/include/Poco.h"
14. **return** 0;
15. }

**特殊的目录和文件**  
               Poco::Path提供了静态函数，用于获取系统中的一些特殊目录和文件  
               1. std::string current()  
               返回当前的工作目录  
               2. std::string home()  
               返回用户的主目录  
               3. std::string temp()  
               返回操作系统的零时目录  
               4. std::string null()  
               返回系统的空目录(e.g., "/dev/null" or "NUL:")  
  
               下面是一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. #include "Poco/Path.h"
2. #include <iostream>
3. **using** Poco::Path;
4. **int** main(**int** argc, **char**\*\* argv)
5. {
6. std::cout
7. << "cwd: " << Path::current() << std::endl
8. << "home: " << Path::home() << std::endl
9. << "temp: " << Path::temp() << std::endl
10. << "null: " << Path::null() << std::endl;
11. **return** 0;
12. }

**路径和环境变量**  
               在配置文件中的路径经常包含了环境变量。在传递此类路径给Poco::Path之间必须对路径进行扩展。  
               对包含环境变量的路径扩展可以使用如下函数  
                               std::string expand(const std::string& path)  
               函数会返回一个对环境变量进行扩充后的路径名。环境变量的格式会根据操作系统有所不同。(e.g., $VAR on Unix, %VAR% on Windows).在Unix上，同样会扩展"~/"为当前用户的主目录。  
  
               下面是一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. #include "Poco/Path.h"
2. **using** Poco::Path;
3. **int** main(**int** argc, **char**\*\* argv)
4. {
5. std::string config("%HOMEDRIVE%%HOMEPATH%\\config.ini");
6. // std::string config("$HOME/config.ini");
7. std::string expConfig(Path::expand(config));
8. **return** 0;
9. }

**文件系统主目录:**  
               void listRoots(std::vector<std::string>& roots)  
               会用所有挂载到文件系统的根目录来填充字符串数组。在windows上，为所有的驱动盘符。在OpenVMS上，是所有挂载的磁盘。在Unix上，为"/"。  
  
**查询文件:**  
               bool find(const std::string& pathList, const std::string& name, Path& path)  
               在指定的目录集(pathList)中搜索指定名称的文件(name)。pathList参数中如果指定了多个查找目录，目录之间必须使用分割符 (Windows平台上为";" , Unix平台上为":")。参数name中可以包含相对路径。如果文件在pathList指定的目录集中存在，找到文件的绝对路径会被放入path参数中，并且函数返回true，否则函数返回false，并且path不改变。  
               这个函数也存在一个使用字符串向量，而不是路径列表的迭代器的变体。定义如下：  
               bool find(StringVec::const\_iterator it, StringVec::const\_iterator end, const std::string& name, Path& path)  
  
               下面是一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. #include "Poco/Path.h"
2. #include "Poco/Environment.h"
3. **using** Poco::Path;
4. **using** Poco::Environment;
5. **int** main(**int** argc, **char**\*\* argv)
6. {
7. std::string shellName("cmd.exe"); // Windows
8. // std::string shellName("sh"); // Unix
9. std::string path(Environment::get("PATH"));
10. Path shellPath;
11. **bool** found = Path::find(path, shellName, shellPath);
12. std::string s(shellPath.toString());
13. **return** 0;
14. }

## 2. Poco::File

**同文件协同工作**  
               1. Poco仅支持与文件元数据协同工作。想要操作文件中的真实数据，需要使用标准库的文件流。  
               2. 使用Poco库，你可以查出一个文件或者目录是否存在，是否可读或可写，文件何时创建和被修改，文件大小等信息。  
               3. 通过Poco库，也可以修改文件属性，重命名文件，拷贝文件和删除文件。  
               4. 通过Poco库，可以创建空文件(原子操作)和目录。  
  
               所有同文件操作相关操作被定义于Poco::File中。为了创建一个Poco::File对象，需要提供一个路径作为参数。这个路径可以使用Poco::Path类，也可以是一个字符串。同时Poco::File也支持创建一个空对象，在稍后的操作中再设置路径。  
               Poco::File支持所有的值语义，也支持所有的关系操作符 (==, !=, <, <=, >, >=)。关系操作符的结果是通过进行简单的文件路径比较而得到的。  
  
               **查询文件属性：**  
               1. bool exists() const  
               如果文件存在返回true，否则为false  
               2. bool canRead() const  
               如果文件可读(用户对文件拥有足够权限)返回true，否则为false  
               3. bool canWrite() const  
               如果文件可写(用户对文件拥有足够权限)返回true，否则为false  
               4. bool canExecute() const  
               如果文件是可执行文件，返回true，否则为false  
               5. bool isFile() const  
               如果文件是常规文件(即不是目录或符号链接)返回为真，否则为false  
               6. bool isLink() const  
               如果文件是符号链接，返回为真，否则为false  
               7. bool isDirectory() const  
               如果文件是目录，返回为真，否则为false  
               8. bool isDevice() const  
               如果文件是设备，返回为真，否则为false  
               9. bool isHidden() const  
               如果文件属性为隐藏，返回为真，否则为false。(在Windows上文件属性为隐藏；Unit上使用"."开头的文件)  
               10. Poco::Timestamp created() const  
               返回文件创建的日期和时间  
               11. Poco::Timestamp getLastModified() const  
               返回最后访问文件的时间和日期  
               12. File::FileSize getSize() const  
               返回文件大小(单位为字节)。在大多数系统中，File::FileSize被定义成一个unsigned 64-bit整数。  
  
**修改文件属性**  
               1. void setLastModified(Poco::Timestamp dateTime)  
               设置最后访问文件的时间  
               2. void setSize(FileSize newSize)  
               设置文件大小(单位为字节)。可用于截断一个文件。  
               3. void setWritable(bool flag = true)  
               如果flag == true，使文件可写；flag == false相当于文件只读  
               4. void setReadOnly(bool flag)  
               同setWritable(!flag)作用相同  
  
**重命名，拷贝和删除**  
               1. void copyTo(const std::string& path) const  
               拷贝文件至指定目录(通常是个文件夹)  
               2. void moveTo(const std::string& path) const  
               拷贝文件至指定目录(通常是个文件夹)并且删除源文件  
               3. void renameTo(const std::string& path)  
               重命名文件  
               4. void remove(bool recursive = false)  
               删除文件。如果文件是个目录，并且recursive == true，则递归的删除所有文件和子目录。  
                 
**创建文件和目录**  
               1. bool createFile()  
               使用原子操作创建一个新的空文件。如果文件被创建返回true，如果文件已存在返回false。如果创建失败，抛出Poco::FileException异常。  
               2. bool createDirectory()  
               创建一个新目录。如果目录被创建，返回true，如果目录已存在，返回false。如果创建失败，抛出Poco::FileException(比如说父目录不存在)。  
               3. void createDirectories()  
               创建目录集。当父目录不存在时，也会同时创建父目录。  
  
**读取目录**  
               1. void list(std::vector<std::string>& files) const  
                   void list(std::vector<File>& files) const  
               会把目录中所有的文件名填入给定的files数组。在其内部，会使用Poco::DirectoryIterator遍历目录  
  
               下面是一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. #include "Poco/File.h"
2. #include "Poco/Path.h"
3. #include <iostream>
4. **using** Poco::File;
5. **using** Poco::Path;
6. **int** main(**int** argc, **char**\*\* argv)
7. {
8. std::string tmpPath(Path::temp());
9. tmpPath.pushDirectory("PocoFileSample");
10. File tmpDir(tmpPath);
11. tmpDir.createDirectories();
12. **bool** exists = tmpDir.exists();
13. **bool** isFile = tmpDir.isFile();
14. **bool** isDir = tmpDir.isDirectory();
15. **bool** canRead = tmpDir.canRead();
16. **bool** canWrite = tmpDir.canWrite();
17. File tmpFile(Path(tmpPath, std::string("PocoFileSample.dat")));
18. **if** (tmpFile.createFile())
19. {
20. tmpFile.setSize(10000);
21. File tmpFile2(Path(tmpPath, std::string("PocoFileSample2.dat")));
22. tmpFile.copyTo(tmpFile2.path());
23. Poco::Timestamp now;
24. tmpFile.setLastModified(now);
25. tmpFile.setReadOnly();
26. canWrite = tmpFile.canWrite();
27. tmpFile.setWriteable();
28. canWrite = tmpFile.canWrite();
29. }
30. std::vector<std::string> files;
31. tmpDir.list(files);
32. std::vector<std::string>::iterator it = files.begin();
33. **for** (; it != files.end(); ++it)
34. {
35. std::cout << \*it << std::endl;
36. }
37. tmpDir.remove(**true**);
38. **return** 0;
39. }

## 3. DirectoryIterator类

               Poco::DirectoryIterator类为读取目录中的内容提供了一个迭代子风格的接口。  
               Poco::DirectoryIterator使用时有如下限制：  
               1. 仅支持前向迭代操作(forward iteration (++))  
               2. 当一个迭代子拷贝自另一个迭代子时，总是指向源迭代子的曾经指向的文件。即使源迭代子已经指向另外一个文件。  
               3. Poco::DirectoryIterator内部维护一个Poco::File和Poco::Path绝对路径的对象。  
  
               下面是一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. #include "Poco/DirectoryIterator.h"
2. #include <iostream>
3. **using** Poco::DirectoryIterator;
4. **using** Poco::Path;
5. **int** main(**int** argc, **char**\*\* argv)
6. {
7. std::string cwd(Path::current());
8. DirectoryIterator it(cwd);
9. DirectoryIterator end;
10. **while** (it != end)
11. {
12. std::cout << it.name();
13. **if** (it->isFile())
14. std::cout << it->getSize();
15. std::cout << std::endl;
16. Path p(it.path());
17. ++it;
18. }
19. **return** 0;
20. }

## 4. Glob类

               Poco::Glob支持同Unix shells类似的通配符匹配。'\*'匹配任何字符串序列，'?'匹配任意单一文件，[SET]匹配任意存在于指定字符集中的单一字符，[!SET]匹配任何不存在于指定字符集中的单一字符。[SET]可以指定字符集的范围和锁包含的字符。例如[123]用来匹配数字1,2,3;[a-zA-Z]匹配任何大写或小写英文字符。Glob类支持用一个反斜杠转义的特殊字符。  
               Poco::Glob可以使用一个pattern，或者一个选项的标志位去创建。GLOB\_DOT\_SPECIAL选项适用于Unix隐藏文件。  
  
**内部函数：**  
               1. bool match(const std::string& subject)  
               如果subject对应的路径符合匹配规则，返回true，否则为false  
               2. void glob(const std::string& pattern, std::set<std::string>& files, int options = 0)  
               void glob(const Path& pattern, std::set<std::string>& files, int options = 0)  
               用符合给定pattern的所有文件填充给定的set集  
  
               下面是一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. #include "Poco/Glob.h"
2. #include <iostream>
3. **using** Poco::Glob;
4. **int** main(**int** argc, **char**\*\* argv)
5. {
6. std::set<std::string> files;
7. Glob::glob("%WINDIR%\\system32\\\*.exe", files);
8. // Glob::glob("/usr/include/\*/\*.h", files);
9. std::set<std::string>::iterator it = files.begin();
10. **for** (; it != files.end(); ++it)
11. {
12. std::cout << \*it << std::endl;
13. }
14. **return** 0;
15. }

## 5. 临时文件

               写程序时，有时候需要用到临时文件，临时文件的使用有一下特点：  
               1. 临时文件只在指定的目录被创建。如Unix系统上的"/tmp/"目录  
               2. 临时文件会自动生成唯一名字  
               3. 临时文件在不再被使用时，需要被删除  
  
               Poco::TemporaryFile中也提供了一个类来实现此功能。

### 5.1 Poco::TemporaryFile类

               1. Poco::TemporaryFile继承自Poco::File  
               2. 构造函数会为临时文件自动生成一个唯一的文件名，以及放置临时文件的操作系统默认目录。当然文件本身并未被创建。  
               3. 如果临时文件被创建，析构函数会删除该文件  
               4. 二选一的，删除动作可以被延迟到程序结束或者不再被使用。  
               5. 任何文件都可以被延迟到程序终止时才被删除。  
  
**成员函数：**  
               1. void keep()  
               停止析构函数中删除文件的动作  
               2. void keepUntilExit()  
               停止析构函数中删除文件的动作，并注册该文件，延迟到程序结束时删除  
               3. static void registerForDeletion(const std::string& path)  
               注册文件，使之在程序结束时删除  
               4. static std::string tempName()  
               为临时文件创建一个唯一的名字  
  
               下面是一个例子：

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8698448)

1. #include "Poco/TemporaryFile.h"
2. #include <fstream>
3. **using** Poco::TemporaryFile;
4. **int** main(**int** argc, **char**\*\* argv)
5. {
6. TemporaryFile tmp;
7. std::ofstream ostr(tmp.path().c\_str());
8. ostr << "Hello, world!" << std::endl;
9. ostr.close();
10. **return** 0;
11. }   大升刷票[www.ruheshuapiao.com](http://www.ruheshuapiao.com) 整理