**链接:** **http://blog.csdn.net/u010312436/article/details/53940309**

**一、简介**

     之所以叫温故而知新，是因为将这两个工具结合起来作为单元**[测试](http://lib.csdn.net/base/softwaretest" \o "软件测试知识库" \t "_blank)**工具的想法在上一个项目中应用了，好像还没有人将这两种工具结合使用，或者没有写成博客供大家参考，现在重新温习下将想法写下来。

    gtest单元测试工具接触过的人都很熟悉了，它是一款google提供的强大的测试框架，测试案例的编写也比较简单，gtest案例的编写可以参考系列博文:http://www.cnblogs.com/coderzh/archive/2009/03/31/1426758.html。

    lcov代码覆盖率统计工具，是gcov的延伸版本，提供程序实际执行的信息（统计某行代码被执行的次数），其基于HTML的输出通过浏览器以清晰的图表形式呈现覆盖率统计结果。locv相关详细介绍可以参考博文：https://my.oschina[**.NET**](http://lib.csdn.net/base/dotnet)/alphajay/blog/33725。

**二、gtest环境的搭建步骤：**

（1）下载源码包搭建：

参考博文：http://www.linuxidc.com/[**Linux**](http://lib.csdn.net/base/linux)/2015-05/116894.htm。

我是按照这篇博客的步骤下载源码将多余的目录删除最后在gtest\_tool目录下只剩下两个核心代码目录：

http://img.blog.csdn.net/20161230114030944?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvdTAxMDMxMjQzNg==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center

（2）直接输入命令安装： sudo apt-get install libgtest-dev

**三、lcov工具的安装：**

（1）下载源码包：http://ltp.sourceforge[**.Net**](http://lib.csdn.net/base/dotnet)/coverage/lcov.**[PHP](http://lib.csdn.net/base/php" \o "PHP知识库" \t "_blank)**

（2）解压：tar xvzf lcov-1.11.tar.gz

（3）cd lcov-1.11

（4）如果是交叉编译移植到实机上需要执行这步：

修改文件：lcov-1.11/bin/genifo  
                    vim lcov-1.11/bin/genifo  
然后将第65行的：our $gcov\_tool = "gcov" 改为自己的交叉编译器的gcov  
  
比如我的交叉编译工具是/usr/local/arm/4.3.2/bin/arm-linux-gcc  
那么就改为：our $gcov\_tool = "/usr/local/arm/4.3.2/bin/arm-linux-gcov"  
  
可以使用：find / -name \*gcov来查找下自己的交叉编译工具在什么目录下

（5）sudo make install

注：除了下载源码包还可以执行下面两条命令安装：

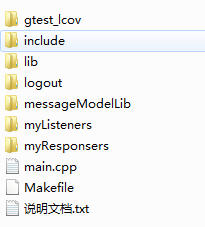
           sudo apt-get install lcov

           sudo apt-get install ggcov

**四、将两者结合使用实例**

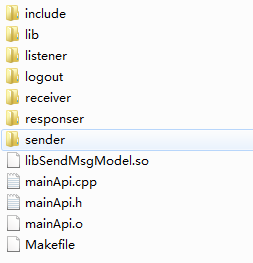
环境搭建好后可以开心得玩耍了 哈哈

（1）首先了解下我需要测试的代码模型：

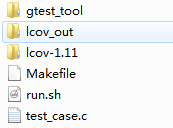


说明：这是我自己写的简单的消息传递及监听模型测试代码。messageModelLib目录是消息传递模型的核心代码，

编译出来一个.so库供该目录的其他模块使用。messageModelLib目录内容如下：



（2）重点是gtest\_lcov目录，该目录是专门用来进行单元测试的，目录内容如下：



说明：gtest\_tool 为gtest源码的删减版，lcov-1.11为lcov源码包编译后的目录，lcov\_out为自己建的目录用来存放lcov工具统计出的结果的输出。test\_case.c为编写的测试用例，内容如下：

**[cpp]** [view plain](http://blog.csdn.net/u010312436/article/details/53940309) [copy](http://blog.csdn.net/u010312436/article/details/53940309)

[在CODE上查看代码片](https://code.csdn.net/snippets/2088028)

1. <span style="font-size:14px;">// Copyright 2005, Google Inc.
2. // All rights reserved.
3. //
4. // Redistribution and use in source and binary forms, with or without
5. // modification, are permitted provided that the following conditions are
6. // met:
7. //
8. //     \* Redistributions of source code must retain the above copyright
9. // notice, this list of conditions and the following disclaimer.
10. //     \* Redistributions in binary form must reproduce the above
11. // copyright notice, this list of conditions and the following disclaimer
12. // in the documentation and/or other materials provided with the
13. // distribution.
14. //     \* Neither the name of Google Inc. nor the names of its
15. // contributors may be used to endorse or promote products derived from
16. // this software without specific prior written permission.
17. //
18. // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
19. // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
20. // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
21. // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
22. // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
23. // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
24. // LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
25. // DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
26. // THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
27. // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
28. // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
30. // A sample program demonstrating using Google C++ testing framework.
31. //
32. // Author: wan@google.com (Zhanyong Wan)

35. // This sample shows how to write a simple unit test for a function,
36. // using Google C++ testing framework.
37. //
38. // Writing a unit test using Google C++ testing framework is easy as 1-2-3:

41. // Step 1. Include necessary header files such that the stuff your
42. // test logic needs is declared.
43. //
44. // Don't forget gtest.h, which declares the testing framework.
46. #include <limits.h>
47. #include "gtest/gtest.h"
49. #include "myComDef.h"
50. #include "responser12.h"
51. #include "listener12.h"
53. extern "C" {
54. #include "log\_out.h"
55. }


59. // Step 2. Use the TEST macro to define your tests.
60. //
61. // TEST has two parameters: the test case name and the test name.
62. // After using the macro, you should define your test logic between a
63. // pair of braces.  You can use a bunch of macros to indicate the
64. // success or failure of a test.  EXPECT\_TRUE and EXPECT\_EQ are
65. // examples of such macros.  For a complete list, see gtest.h.
66. //
67. // <TechnicalDetails>
68. //
69. // In Google Test, tests are grouped into test cases.  This is how we
70. // keep test code organized.  You should put logically related tests
71. // into the same test case.
72. //
73. // The test case name and the test name should both be valid C++
74. // identifiers.  And you should not use underscore (\_) in the names.
75. //
76. // Google Test guarantees that each test you define is run exactly
77. // once, but it makes no guarantee on the order the tests are
78. // executed.  Therefore, you should write your tests in such a way
79. // that their results don't depend on their order.
80. //
81. // </TechnicalDetails>

84. // Tests Factorial().
86. // Tests factorial of negative numbers.
88. static responser1 res1\_instance;
89. static responser2 res2\_instance;
91. static listener1 lis1\_instance;
92. static listener2 lis2\_instance;

95. TEST(apl\_registResponserTest, isTrue)
96. {
97. ASSERT\_EQ(TRUE, apl\_registResponser(RES1, RES\_MID, &res1\_instance));
98. ASSERT\_EQ(TRUE, apl\_registResponser(RES2, RES\_HIGH, &res2\_instance));
99. }

102. TEST(apl\_registListenerTest, isTrue)
103. {
104. ASSERT\_EQ(TRUE, apl\_registListener(RES1, &res1\_instance, &lis1\_instance));
105. ASSERT\_EQ(TRUE, apl\_registListener(RES2, &res2\_instance, &lis2\_instance));
106. }

109. TEST(loop\_send\_test, isRight)
110. {
111. int n = 0;
112. while(n < 10)
113. {
114. MSG\_INFO msgInfo;
115. char msgData[128];
117. msgInfo.resType = RES1;  //设置动作对象
118. msgInfo.msg.eventID = R1\_FUN1; //设置应该做什么动作
119. EXPECT\_EQ(TRUE, apl\_sendMessage(RES1, R1\_FUN1, msgData)); //发送开始消息及动作参数
121. //apl\_unRegistResponser(RES1);
123. msgInfo.msg.eventID = R1\_FUN2;
124. int a = 10, b = 122;
125. apl\_msgPacker(msgData,sizeof(int), &a,sizeof(int), &b, -1);
126. EXPECT\_EQ(TRUE, apl\_sendMessage(RES1, R1\_FUN2, msgData));
128. msgInfo.resType = RES2;
129. msgInfo.msg.eventID = R2\_FUN1;
130. double c = 4.23, d = 2.32;
131. apl\_msgPacker(msgData,sizeof(double), &c,sizeof(double), &d, -1);
132. EXPECT\_EQ(TRUE, apl\_sendMessage(RES2, R2\_FUN1, msgData));
134. msgInfo.msg.eventID = R2\_FUN2;
135. a = 20;
136. b = 30;
137. apl\_msgPacker(msgData,sizeof(int), &a,sizeof(int), &b, -1);
138. EXPECT\_EQ(TRUE, apl\_sendMessage(RES2, R2\_FUN2, msgData));
140. sleep(1);
141. ++n;
142. }
143. }

146. // Step 3. Call RUN\_ALL\_TESTS() in main().
147. //
148. // We do this by linking in src/gtest\_main.cc file, which consists of
149. // a main() function which calls RUN\_ALL\_TESTS() for us.
150. //
151. // This runs all the tests you've defined, prints the result, and
152. // returns 0 if successful, or 1 otherwise.
153. //
154. // Did you notice that we didn't register the tests?  The
155. // RUN\_ALL\_TESTS() macro magically knows about all the tests we
156. // defined.  Isn't this convenient?
157. </span>

**主要测试三个case，消息回应者、对应监听器的注册和消息的传递和监听。**

**编写好test\_case.c以后文件以后，关键的关键就是makefile的编写了，我的makefile是gtest源码包example测试makefile基础上修改的，内容如下：**

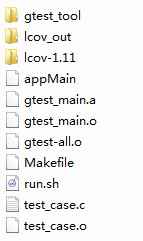
**[cpp]** [view plain](http://blog.csdn.net/u010312436/article/details/53940309) [copy](http://blog.csdn.net/u010312436/article/details/53940309)

[在CODE上查看代码片](https://code.csdn.net/snippets/2088028)

1. <span style="font-size:14px;"># A sample Makefile for building Google Test and using it in user
2. # tests.  Please tweak it to suit your environment and project.  You
3. # may want to move it to your project's root directory.
4. #
5. # SYNOPSIS:
6. #
7. #   make [all]  - makes everything.
8. #   make TARGET - makes the given target.
9. #   make clean  - removes all files generated by make.
11. # Please tweak the following variable definitions as needed by your
12. # project, except GTEST\_HEADERS, which you can use in your own targets
13. # but shouldn't modify.
15. # Points to the root of Google Test, relative to where this file is.
16. # Remember to tweak this if you move this file.
17. <span style="color:#3333FF;">GTEST\_DIR = ./gtest\_tool #gtest源码目录</span>
18. XX = g++
19. CC = gcc
20. # Where to find user code.
21. <span style="color:#FF0000;">USER\_DIR = ..  #需要单元测试的代码目录</span>
23. # Flags passed to the preprocessor.
24. # Set Google Test's header directory as a system directory, such that
25. # the compiler doesn't generate warnings in Google Test headers.
26. CPPFLAGS += -isystem $(GTEST\_DIR)/include
28. # Flags passed to the C++ compiler.
29. CXXFLAGS += -g -Wall -Wextra -pthread
30. CXXFLAGS2 += -g -Wall -Wextra -pthread <span style="color:#CC0000;">-fprofile-arcs -ftest-coverage</span>
31. # All tests produced by this Makefile.  Remember to add new tests you
32. # created to the list.
33. TARGET = appMain
35. # the link library you should change according to your need
36. LINK\_LIB = -L$(USER\_DIR)/lib -lSendMsgModel -lpthread -lrt
38. # All Google Test headers.  Usually you shouldn't change this
39. # definition.
40. GTEST\_HEADERS = $(GTEST\_DIR)/include/gtest/\*.h \
41. $(GTEST\_DIR)/include/gtest/internal/\*.h
43. # House-keeping build targets.
45. all : $(TARGET)
47. # Builds gtest.a and gtest\_main.a.
49. # Usually you shouldn't tweak such internal variables, indicated by a
50. # trailing \_.
51. GTEST\_SRCS\_ = $(GTEST\_DIR)/src/\*.cc $(GTEST\_DIR)/src/\*.h $(GTEST\_HEADERS)
53. # For simplicity and to avoid depending on Google Test's
54. # implementation details, the dependencies specified below are
55. # conservative and not optimized.  This is fine as Google Test
56. # compiles fast and for ordinary users its source rarely changes.
57. gtest-all.o : $(GTEST\_SRCS\_)
58. g++ $(CPPFLAGS) -I$(GTEST\_DIR) -c \
59. $(GTEST\_DIR)/src/gtest-all.cc
61. gtest\_main.o : $(GTEST\_SRCS\_)
62. g++ $(CPPFLAGS) -I$(GTEST\_DIR) -c \
63. $(GTEST\_DIR)/src/gtest\_main.cc
65. gtest.a : gtest-all.o
66. $(AR) $(ARFLAGS) $@ $^
68. gtest\_main.a : gtest-all.o gtest\_main.o
69. $(AR) $(ARFLAGS) $@ $^
71. # Builds a sample test.  A test should link with either gtest.a or
72. # gtest\_main.a, depending on whether it defines its own main()
73. # function.
75. XX\_SOURCES = $(wildcard $(USER\_DIR)/myListeners/\*.cpp $(USER\_DIR)/myResponsers/\*.cpp)
76. CC\_SOURCES = $(wildcard $(USER\_DIR)/logout/\*.c)
78. XX\_OBJECTS = $(patsubst %.cpp,%.o,$(XX\_SOURCES))
79. CC\_OBJECTS = $(patsubst %.c,%.o,$(CC\_SOURCES))
81. INCLUDE\_DIRS = -I$(USER\_DIR)/include -I$(USER\_DIR)/myListeners -I$(USER\_DIR)/myResponsers -I$(USER\_DIR)/logout
83. <span style="color:#CC33CC;">TEST\_CASE\_O = ./test\_case.o</span>
85. $(TEST\_CASE\_O) : %.o : %.c
86. $(XX) -c $(CPPFLAGS) $< -o $@ $(INCLUDE\_DIRS)
88. $(CC\_OBJECTS) : %.o : %.c
89. $(CC) -c $(<span style="color:#FF0000;">CXXFLAGS2</span>) $< -o $@ $(INCLUDE\_DIRS)  <em><span style="color:#6633FF;">#需要用lcov查看哪个文件的代码覆盖率，编译的时候就加上-fprofile-arcs -ftest-coverage编译选项</span></em>
91. $(XX\_OBJECTS) : %.o : %.cpp
92. $(XX) -c $(<span style="color:#FF0000;">CXXFLAGS2</span>) $< -o $@ $(INCLUDE\_DIRS)
94. $(TARGET) : $(XX\_OBJECTS) $(CC\_OBJECTS) <span style="color:#CC33CC;">$(TEST\_CASE\_O) gtest\_main.a</span>
95. $(XX) $(<span style="color:#FF0000;">CXXFLAGS2</span>) $^ -o $@ $(LINK\_LIB)
97. #删除代码目录的 \*.gcda,\*.gcno和\*.o文件
98. SUBDIRS = $(USER\_DIR)/myListeners $(USER\_DIR)/myResponsers $(USER\_DIR)/logout
99. GCDA\_FILES = $(foreach dir, $(SUBDIRS), $(wildcard $(dir)/\*.gcda))
100. GCNO\_FILES = $(foreach dir, $(SUBDIRS), $(wildcard $(dir)/\*.gcno))
101. OBJS\_FILES = $(foreach dir, $(SUBDIRS), $(wildcard $(dir)/\*.o))
103. clean :
104. rm -rf $(TARGET) gtest.a gtest\_main.a \*.o \*.gcno \*.gcda \*.info lcov\_out/\* \
105. $(GCDA\_FILES) $(GCNO\_FILES) $(OBJS\_FILES)</span>

其中带颜色的部分是需要注意的。

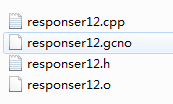
（3）在gtest\_lcov目录下执行 make 命令后生成：



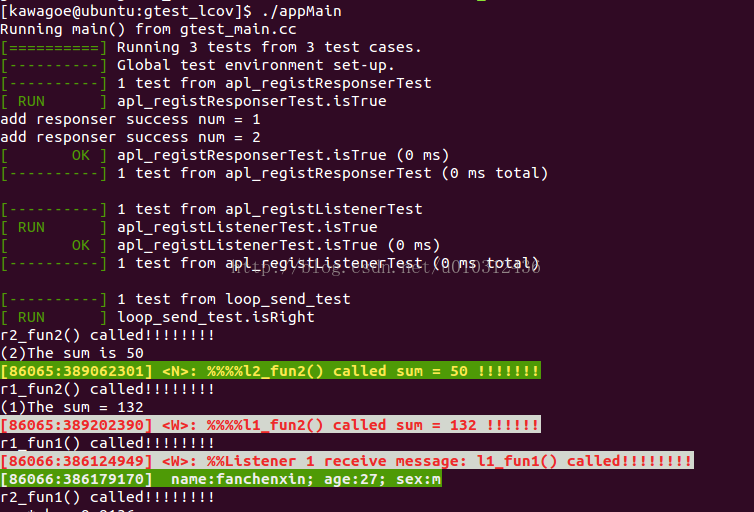
**[cpp]** [view plain](http://blog.csdn.net/u010312436/article/details/53940309) [copy](http://blog.csdn.net/u010312436/article/details/53940309)

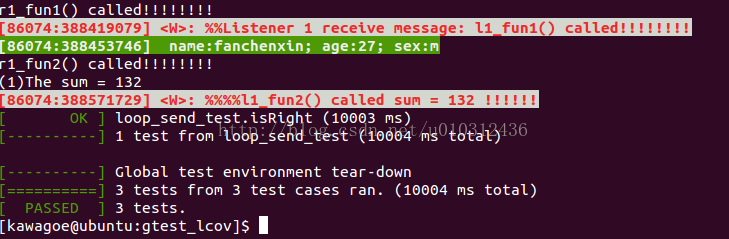
[在CODE上查看代码片](https://code.csdn.net/snippets/2088028)

1. <span style="font-size:14px;"><span style="color:#CC0000;">加 -fprofile-arcs -ftest-coverage 编译选项编译的文件会生成相应的.gcno文件，
2. 因此测试代码目录下的三个目录logout, myResponsers和myListeners目录下生成\*.gcno文件如下：</span></span>



（4）执行编译生成的可执行文件 appMain, 就会看到gtest的测试结果：



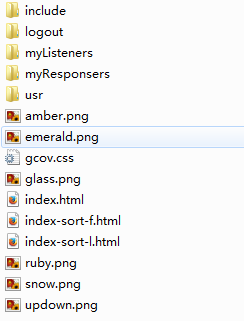


执行完后也会在生成gcon文件的目录下生成gcda文件。

（5）用lcov工具生成可视的图表，并用浏览器查看。

##  参数-d 后面跟的目录是.gcon .gcda 所在的根目录  
**./lcov-1.11/bin/lcov -d ../ -t 'appMain' -o 'appMain.info' -b . -c  
./lcov-1.11/bin/genhtml appMain.info --quiet --output-directory lcov\_out --title "appMain"  
firefox ./lcov\_out/index.html**

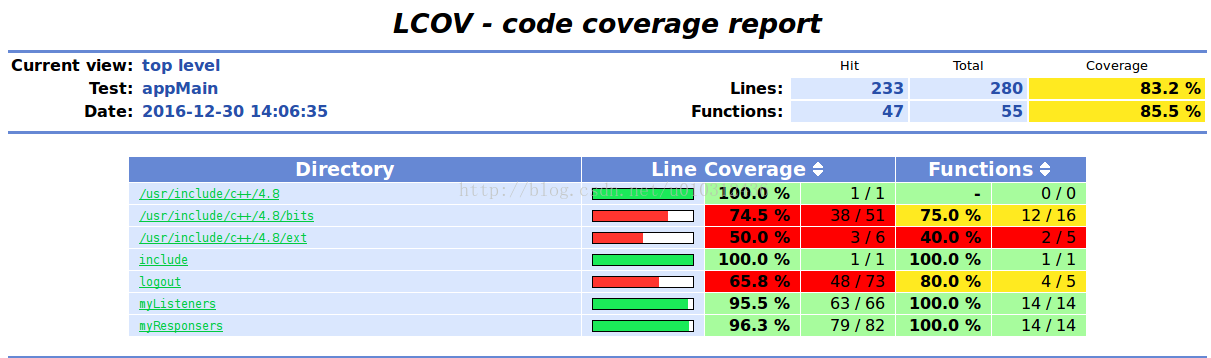
lcov\_out目录下生成的内容：

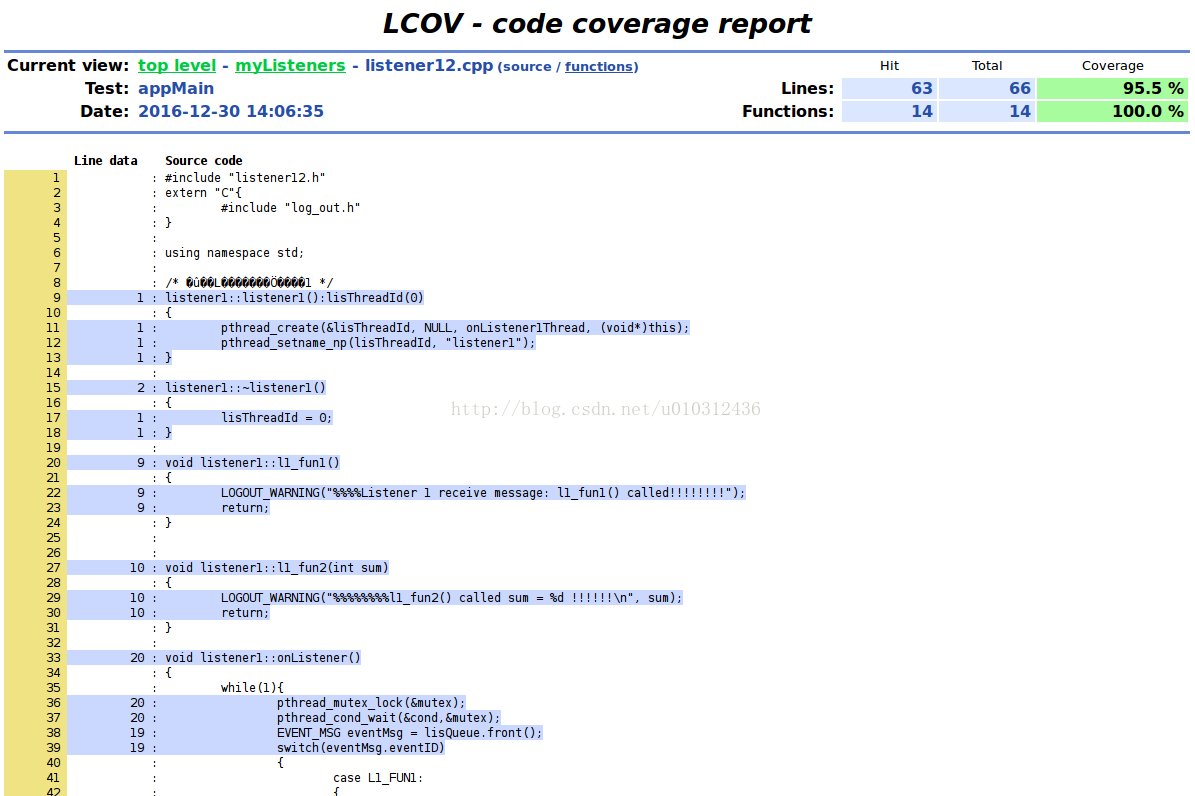


（6）最后把测试的命令（1~5）命令写到run.sh脚本里，执行./run.sh就ok了。

#! /bin/bash  
cd ..  
make clean  
cd ./gtest\_lcov  
make clean  
sudo rm -rf ./lcov\_out/\*  
make  
./appMain  
  
./lcov-1.11/bin/lcov -d ../ -t 'appMain' -o 'appMain.info' -b . -c  
./lcov-1.11/bin/genhtml appMain.info --quiet --output-directory lcov\_out --title "appMain"  
firefox ./lcov\_out/index.html

（7）覆盖率统计图表：





至此，gtest单元测试工具和lcov覆盖率统计工具的结合使用介绍完毕，共同学习进步。