log4c安装使用教程

一、在线安装及使用

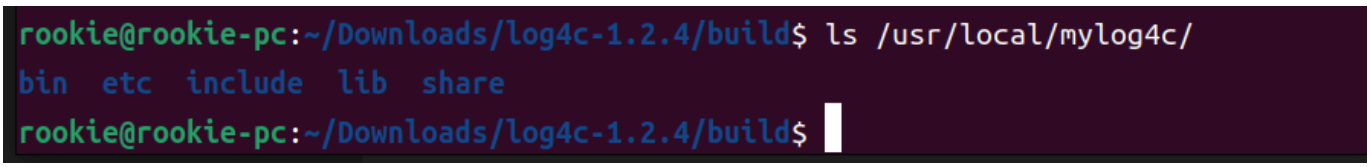
安装

1. 下载源码，地址：GitHub - bmanojlovic/log4c: Fork of Log4c project
2. 使用以下命令进行解压、配置、安装

```
tar -vxf log4c-1.2.4.tar.gz
cd log4c-1.2.4
mkdir build
cd build
../configure --without-expat --prefix=/usr/local/mylog4c
make
make install
```

注意：上述prefix指定安装目录，根据自己需求进行更改

安装成功如下：



使用

示例工程采用CMake进行工程管理，**CMake文件**如下：

```
cmake_minimum_required(VERSION 3.25)
project(test_log4c C)

set(CMAKE_C_STANDARD 11)

# log4c
set(3RDLIB_PATH /usr/local/mylog4c)
list(APPEND 3RD_LIBS_INCLUDE ${3RDLIB_PATH}/include)
list(APPEND 3RD_LIBS_INCLUDE ${3RDLIB_PATH}/include/log4c)
include_directories(${3RD_LIBS_INCLUDE})
link_directories(${3RDLIB_PATH}/lib)
link_libraries(log4c)
add_executable(test_log4c main.c)
```

C程序如下：

```
#include <stdio.h>
#include <log4c.h>

int main() {

    log4c_category_t* mycat = NULL;
    if (log4c_init()){
        printf("log4c_init() failed\n");
    }else{
        mycat = log4c_category_get("six13log.log.app"); //xml配置文件中category标签页的名字

        log4c_category_log(mycat, LOG4C_PRIORITY_ERROR, "Hello World!");

        /* Explicitly call the log4c cleanup routine */
        if (log4c_fini()){
            printf("log4c_fini() failed\n");
        }
    }
    return 0;
}
```

程序说明：

在运行时我们需要**在可执行文件同级目录下创建一个名为log4crc的文件**，该文件用于配置log输出情况，文件格式如下：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE log4c SYSTEM "">

<log4c version="1.2.4">

    <config>
        <bufsize>0</bufsize>
        <debug level="2"/>
        <nocleanup>0</nocleanup>
        <reread>1</reread>
    </config>

    <category name="root" priority="notice"/>
    <category name="six13log.log" priority="error" appender="stdout" />
    <category name="six13log.log.app.application2" priority="debug" appender="cat_stderr" />
    <category name="six13log.log.app.application3" priority="debug" appender="user_stderr" />
    <category name="six13log.log.app" priority="debug" appender="myrollingfileappender" />

    <rollingpolicy name="myrollingpolicy" type="sizewin" maxsize="1024" maxnum="10" />
    <appender name="myrollingfileappender" type="rollingfile" logdir="." prefix="myprefix"
layout="dated" rollingpolicy="myrollingpolicy" />

    <appender name="stdout" type="stream" layout="basic"/>
    <appender name="stderr" type="stream" layout="dated"/>
    <appender name="syslog" type="syslog" layout="basic"/>
    <appender name="s13file" type="s13_file" layout="basic"/>
    <appender name="plain_stderr" type="s13_stderr" layout="none"/>
    <appender name="cat_stderr" type="s13_stderr" layout="catlayout"/>
    <appender name="xml_stderr" type="s13_stderr" layout="xmllayout"/>
    <appender name="user_stderr" type="s13_stderr" layout="userlayout"/>

    <layout name="basic" type="basic"/>
    <layout name="dated" type="dated"/>
    <layout name="catlayout" type="s13_cat"/>
    <layout name="xmllayout" type="s13_xml"/>
    <layout name="none" type="s13_none"/>
    <layout name="userlayout" type="s13_userloc"/>

</log4c>
```

配置xml文件常见标签说明：

```
<rollingpolicy name="myrollingpolicy" type="sizewin" maxsize="1024" maxnum="10" />
```

- `name="myrollingpolicy"`：指定了这个滚动策略的名称为"myrollingpolicy"，在配置文件中可以通过这个名称引用该策略。
- `type="sizewin"`：指定了滚动策略的类型为"sizewin"，表示按照文件大小来滚动日志。
- `maxsize="1024"`：指定了单个日志文件的最大大小为1024（通常表示1024字节，也就是1KB）。当日志文件大小达到这个阈值时，系统会触发滚动操作。
- `maxnum="10"`：指定了保留的日志文件数量为10。这意味着系统会保留最新的10个滚动后的日志文件，旧的日志文件会被删除或者归档。

综上所述，这个滚动策略的作用是在单个日志文件达到1KB时触发滚动操作，并且保留最新的10个滚动后的日志文件。

```
<appender name="myrollingfileappender" type="rollingfile" logdir="." prefix="myprefix"
layout="dated" rollingpolicy="myrollingpolicy" />
```

这段XML代码是关于日志文件附加器（appender）的配置，它指定了如何将日志消息写入文件，并且使用了之前定义的滚动策略。

以下是对其中各部分的说明：

- `<appender>`：这是一个日志文件附加器的配置块，用于指定日志消息的输出目标和方式。
- `name="myrollingfileappender"`：指定了这个日志文件附加器的名称为"myrollingfileappender"，在配置文件中可以通过这个名称引用该附加器。
- `type="rollingfile"`：指定了附加器的类型为"rollingfile"，表示日志消息将会写入到一个滚动文件中。
- `logdir="."`：指定了日志文件的存储目录为当前目录（"."表示当前目录）。这表示日志文件会被存储在配置文件所在的目录中。
- `prefix="myprefix"`：指定了日志文件名的前缀为"myprefix"。这意味着生成的日志文件名会以"myprefix"开头。
- `layout="dated"`：指定了日志消息的布局格式为"dated"，通常是指按日期格式化日志消息。
- `rollingpolicy="myrollingpolicy"`：指定了使用之前定义的名为"myrollingpolicy"的滚动策略来管理日志文件的滚动。

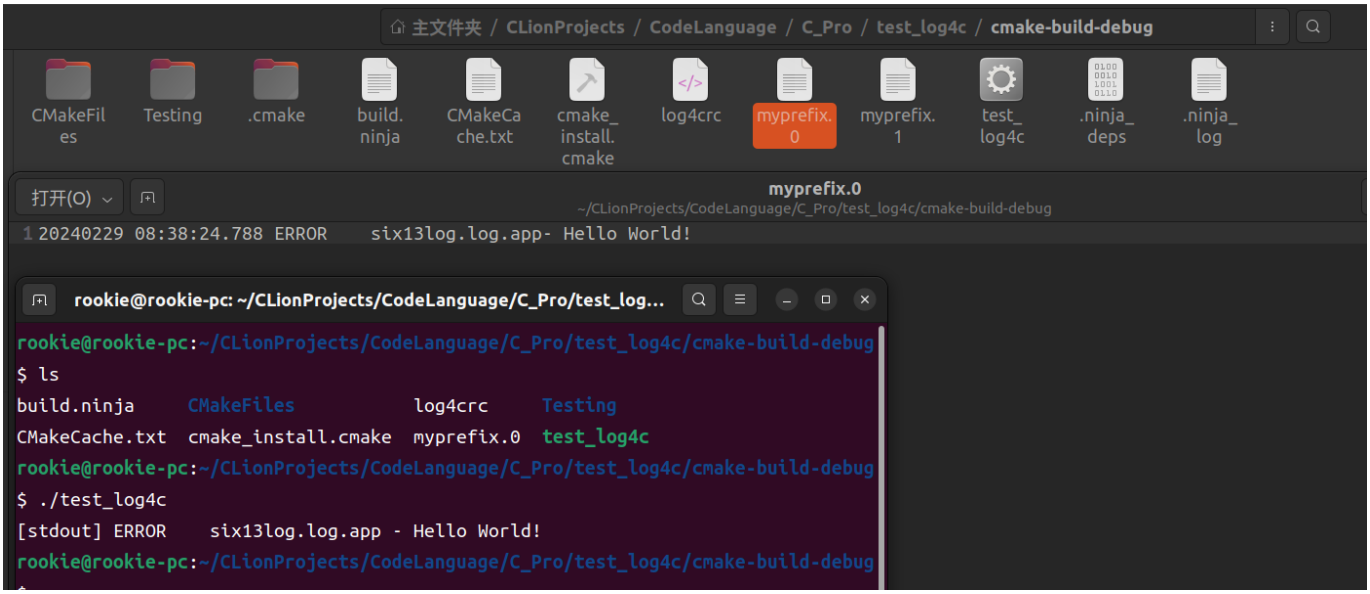
综上所述，这个日志文件附加器的作用是将日志消息写入滚动文件中，文件存储在当前目录下，文件名以"myprefix"开头，日志消息按照日期格式化，并且使用了之前定义的名

为"myrollingpolicy"的滚动策略来管理日志文件的滚动。

以上述C程序为例，我们使用的category是"six13log.log.app"，该category属性对应的appender为"myrollingfileappender"，我们查看名为"myrollingfileappender"的appender标签发现它会循环滚动保存文件。

同时我们发现xml文件中有一个名为"six13log.log"的category标签。这就不得不提log4c的category具有父子关系（如上述xml文件："six13log.log.app"的父亲是"six13log.log"）。

当我们调用名为"six13log.log.app"的category时也会调用其父category的打印，所用log不仅会存入文件还会打印在终端，输出效果如下图：



当我们希望log只打印到终端（或者文件）时，我们需要单独创建一个category，不要包含父子关系，例如下图：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE log4c SYSTEM "">

<log4c version="1.2.4">

    <config>
        <bufsize>0</bufsize>
        <debug level="2"/>
        <nocleanup>0</nocleanup>
        <reread>1</reread>
    </config>

    <category name="root" priority="notice"/>
    <category name="six13log.log" priority="error" appender="stdout" />
    <category name="output_file" priority="debug" appender="myrollingfileappender" />

    <rollingpolicy name="myrollingpolicy" type="sizewin" maxsize="1024" maxnum="10" />
    <appender name="myrollingfileappender" type="rollingfile" logdir="." prefix="myprefix"
layout="dated" rollingpolicy="myrollingpolicy" />
    <appender name="stdout" type="stream" layout="basic"/>

    <layout name="basic" type="basic"/>
    <layout name="dated" type="dated"/>

</log4c>
```

二、离线源码编译与使用

针对某些场景下，我们不太希望往系统中安装log4c的库，而更希望将log4c以源码的方式移植到程序中，这时就不能按照章节一进行安装，为此我提供一份移植好的源码。

CMake文件如下：

```
cmake_minimum_required(VERSION 3.25)
project(LOG4C C)

set(CMAKE_C_STANDARD 11)

# 增加宏定义
add_compile_definitions(HAVE_CONFIG_H)
# main.c路径
set(MAIN_DIR ${CMAKE_CURRENT_SOURCE_DIR}/main.c)
# 头文件路径
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/src/log4c)
include_directories(${CMAKE_CURRENT_SOURCE_DIR}/src/sd)
# 源文件路径
aux_source_directory(${CMAKE_CURRENT_SOURCE_DIR}/src/log4c SRC_LIST)
aux_source_directory(${CMAKE_CURRENT_SOURCE_DIR}/src/sd SRC_LIST2)
# 添加源文件
set(SRC_ALL_LIST ${SRC_LIST}  ${SRC_LIST2} ${MAIN_DIR})
add_executable(LOG4C ${SRC_ALL_LIST})
## 链接第三方库(使用第三方expat解析xml文件时开启，使用自带xml解析不需要开启，目前的源码使用自带xml解析)
#find_package(EXPAT REQUIRED)
#if(EXPAT_FOUND)
#    include_directories(${EXPAT_INCLUDE_DIRS})
#    target_link_libraries(LOG4C ${EXPAT_LIBRARIES})
#endif(EXPAT_FOUND)
```

C程序可以使用章节一种的例程，xml文件同样也通用。

三、离线源码部分更改说明

我们侧重讲一下源码中一些比较重要的更改。

1. log4crc搜索路径，init.c中的recfiles数组，安装从上往下的顺序遍历，找到指定文件后不再遍历后续文件，相关代码如下：

```
        //搜索文件路径
static rcfile_t rcfiles[] = {
    { "/log4crc" },
    { "/home/rookie/log4crc" },
    { "./log4crc" }
};

static const int nrcfiles = sizeof(rcfiles) / sizeof(rcfiles[0]);

.......省略.....

for (i = 0; i < nrcfiles; i++) {
     sd_debug("checking for conf file at '%s'", rcfiles[i].name);
     if (SD_ACCESS_READ(rcfiles[i].name)) continue;
     if (SD_STAT_CTIME(rcfiles[i].name,&rcfiles[i].ctime) != 0)
     sd_error("sd_stat_ctime %s failed", rcfiles[i].name);
     rcfiles[i].exists=1;
     if (log4c_load(rcfiles[i].name) == -1) {
          sd_error("loading %s failed", rcfiles[i].name);
          ret = -1;
     }
     else {
          sd_debug("loading %s succeeded", rcfiles[i].name);
          ret = 0;
          break;
     }
}
```

2. 源码config.h中的VERSION要与xml中的版本相同

```
/* Define to 1 if your <sys/time.h> declares `struct tm'. */
/* #undef TM_IN_SYS_TIME */

/* Version number of package */
#define VERSION "1.2.4"

/* build log4c with initialization constructors */
/* #undef WITH_CONSTRUCTORS */
```

打开(O)　log4crc
~/CLionProjects/CodeLanguage/C_Pro/LOG4C/cmake-build-debug

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE log4c SYSTEM "">

<log4c version="1.2.4">

        <config>
                <bufsize>0</bufsize>
                <debug level="2"/>
                <nocleanup>0</nocleanup>
                <reread>1</reread>
        </config>
```

3. 打印输出文本格式，主要更改layout_type_xxx.c中的xxx_format函数，如下图：

LOG4C – layout_type_basic.c

文件(F) 编辑(E) 视图(V) 导航(N) 代码(C) 重构(R) 构建(B) 运行(U) 工具(T) Git(G) 窗口(W) 帮助(H)

LOG4C 〉 src 〉 log4c 〉 layout_type_basic.c

项目

defs.h
init.c
init.h
layout.c
layout.h
layout_type_basic.c
layout_type_basic.h
layout_type_basic_r.c
layout_type_basic_r.h
layout_type_dated.c
layout_type_dated.h
layout_type_dated_local.c
layout_type_dated_local.h
layout_type_dated_local_r.c
layout_type_dated_local_r.h
layout_type_dated_r.c
layout_type_dated_r.h
location_info.h
log4c.h
logging_event.c
logging_event.h
priority.c
priority.h
rc.c
rc.h

init.c × rc.h × rc.c × version.h × config.h × layout_type_basic.c × version.c × main.c × category

```
#include <stdio.h>

/*******************************************************************************/
static const char* basic_format(
    const log4c_layout_t*       a_layout,
    const log4c_logging_event_t*    a_event)
{
    static char buffer[1024];

    snprintf( s: buffer,  maxlen: sizeof(buffer),  format: "%-8s %s - %s\n",
        log4c_priority_to_string( a_priority: a_event->evt_priority),
        a_event->evt_category, a_event->evt_msg);

    return buffer;
}

/*******************************************************************************/
const log4c_layout_type_t log4c_layout_type_basic = {
    .name: "basic",
    .format: basic_format,
};
```

四、其他学习资料

关于log4c设计原理可以参考学习博客：沉淀之log4c库 - cfzhang - 博客园