1.MQTT 简介

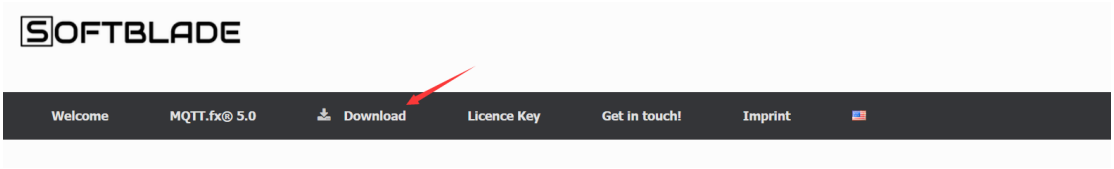
物联网曾被认为是继计算机、互联网之后，信息技术行业的第三次浪潮。随着基础通讯设施的不断完善，尤其是 5G 的出现，进一步降低了万物互联的门槛和成本。物联网本身也是 AI 和区块链应用很好的落地场景之一，各大云服务商也在纷纷上架物联网平台和服务。

物联网通讯是物联网的一个核心内容，目前物联网的通讯协议并没有一个统一的标准，比较常见的有 MQTT、CoAP、DDS、XMPP 等，在这其中，MQTT（消息队列遥测传输协议）应该是应用最广泛的标准之一。目前，MQTT 已逐渐成为 IoT 领域最热门的协议，也是国内外各大物联网平台最主流的传输协议，阿里云 IoT 物联网平台很多设备都是通过 MQTT 接入。

我们将向您讲解 MQTT 协议的应用，一起在开发板上实现一个简单地物联网小项目，实现远程控制开发板上的外设，譬如 LED、蜂鸣器；亦或者远程获取开发板运行的状态信息。

2.下载安装 MQTT.fx 软件

想要将电脑作为 MQTT 客户端，我们需要在电脑上安装一个 MQTT 客户端软件，MQTT 客户端软件很多，这里笔者推荐 MQTT.fx 这款软件，这款软件是目前主流的 MQTT 桌面客户端，它支持 Windows、Mac、Linux 等多种操作系统，它的官网是 http://mqttfx.jensd.de/。进入到官网下载 MQTT.fx 软件，如下所示：



这里是笔者下载好的 MQTT.fx 软件：

若是无法在官网下载可以去 github 上看看，上面有很多相关的开源软件。

3.MQTT 服务端测试

除了自己搭建服务器之外，我们还可以使用现有的 MQTT 服务器，譬如阿里云、百度云、华为云等提供的 MQTT 服务，不过这些大平台貌似都是收费的；对于我们学习测试来说非常不友好，那既然如此我们还有别的选择吗？当然有，我们可以使用公用 MQTT 服务器，这些服务器都是免费供大家学习测试使用的，需要注意的是这些公用 MQTT 服务器仅用于学习测试，不可拿来商用！以下给大家列举了一些公用 MQTT 服务器：

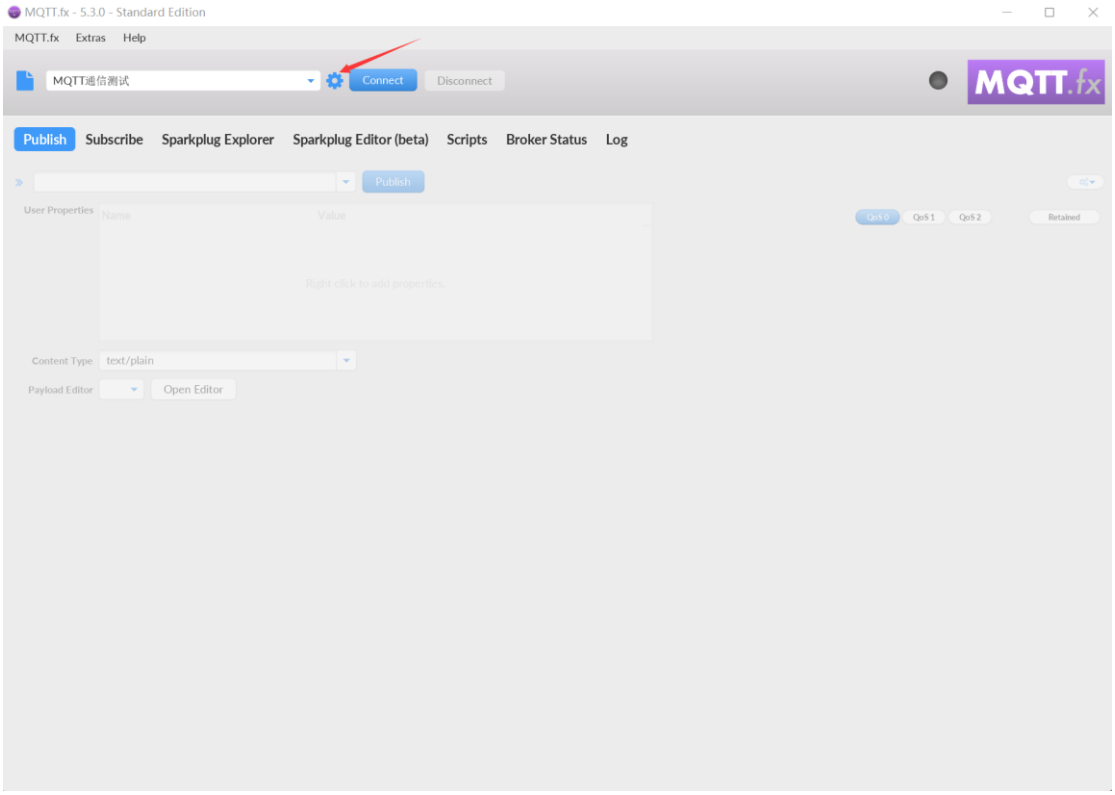
然也物联(国内)

官网地址：http://www.ranye-iot.net
MQTT 服务器地址：test.ranye-iot.net
TCP 端口：1883
TCP/TLS 端口：8883

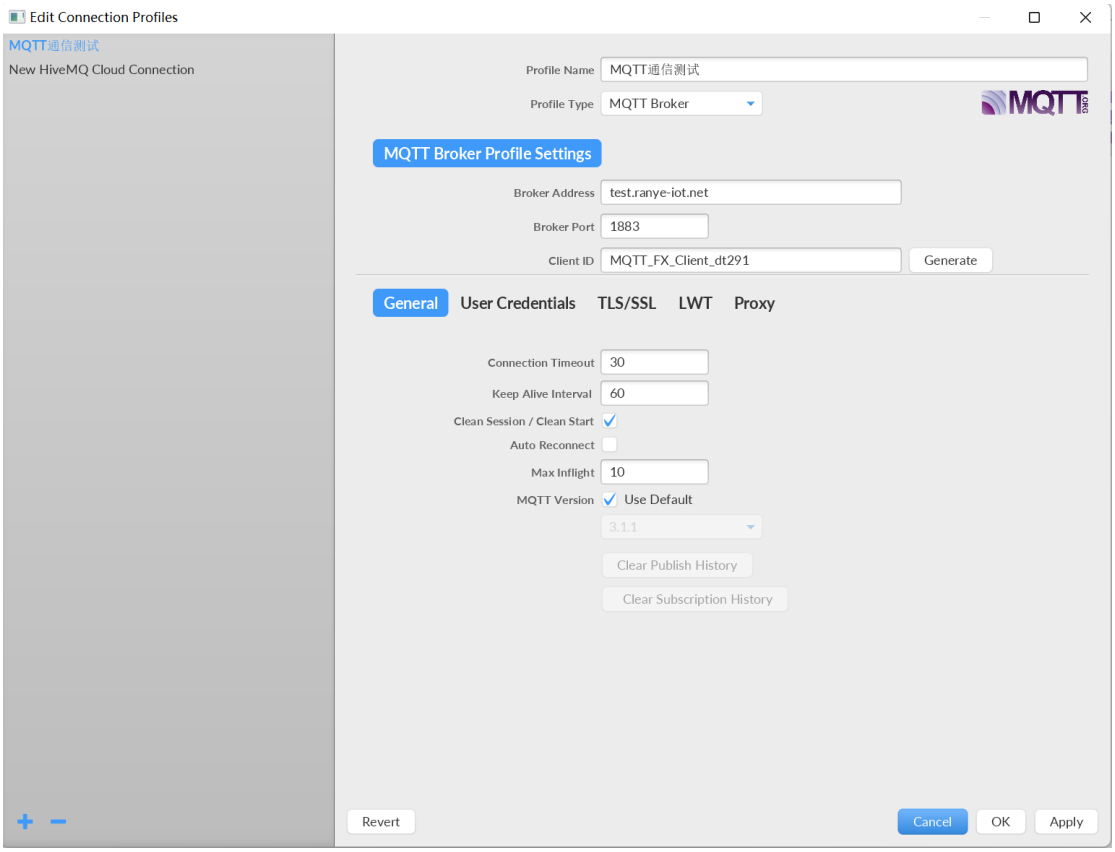
通信猫（国内）

MQTT 服务器地址：mq.tongxinmao.com
TCP 端口：1883

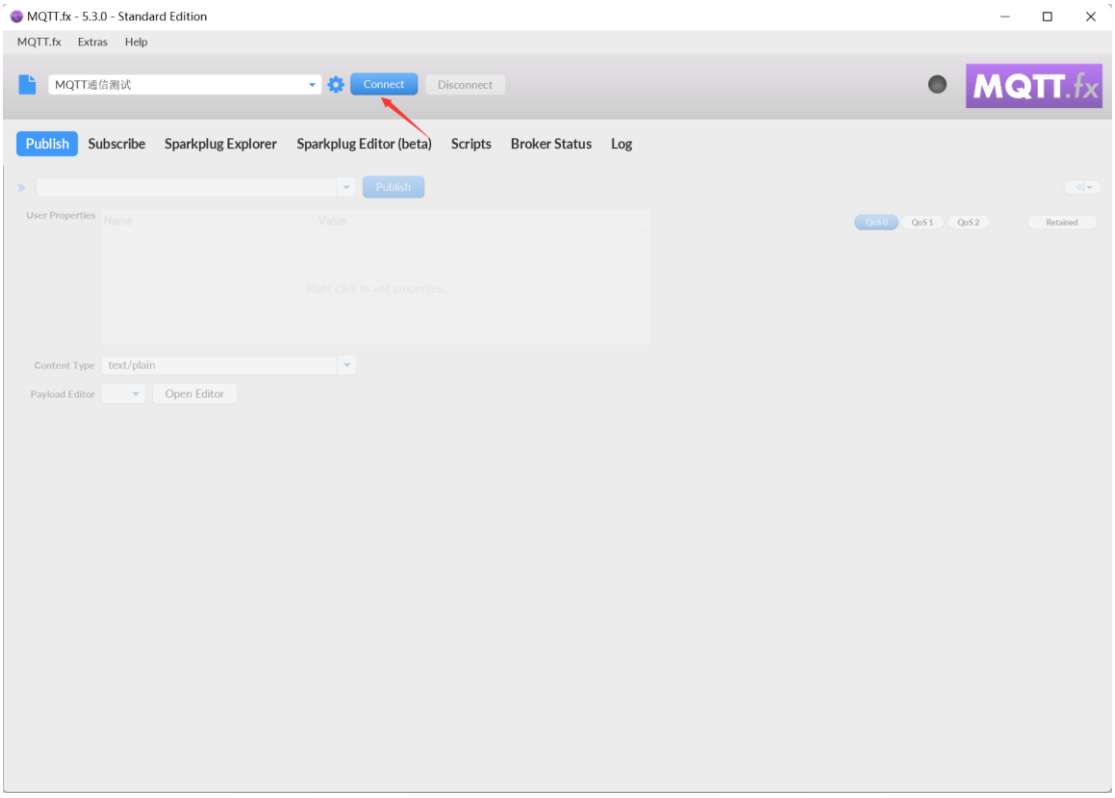
现在我们要动手进行 MQTT 通信测试了，首先打开之前安装的 MQTT 客户端软件 MQTT.fx，点击小齿轮：



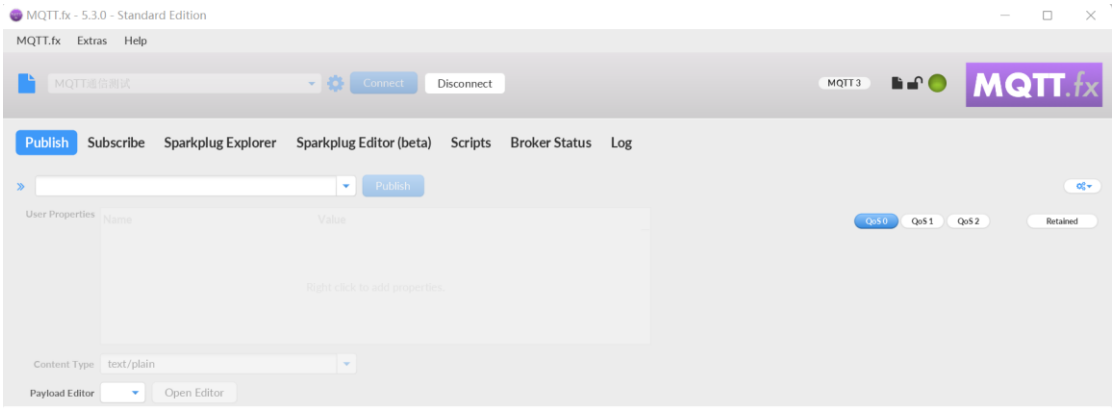
配置好相应的参数：



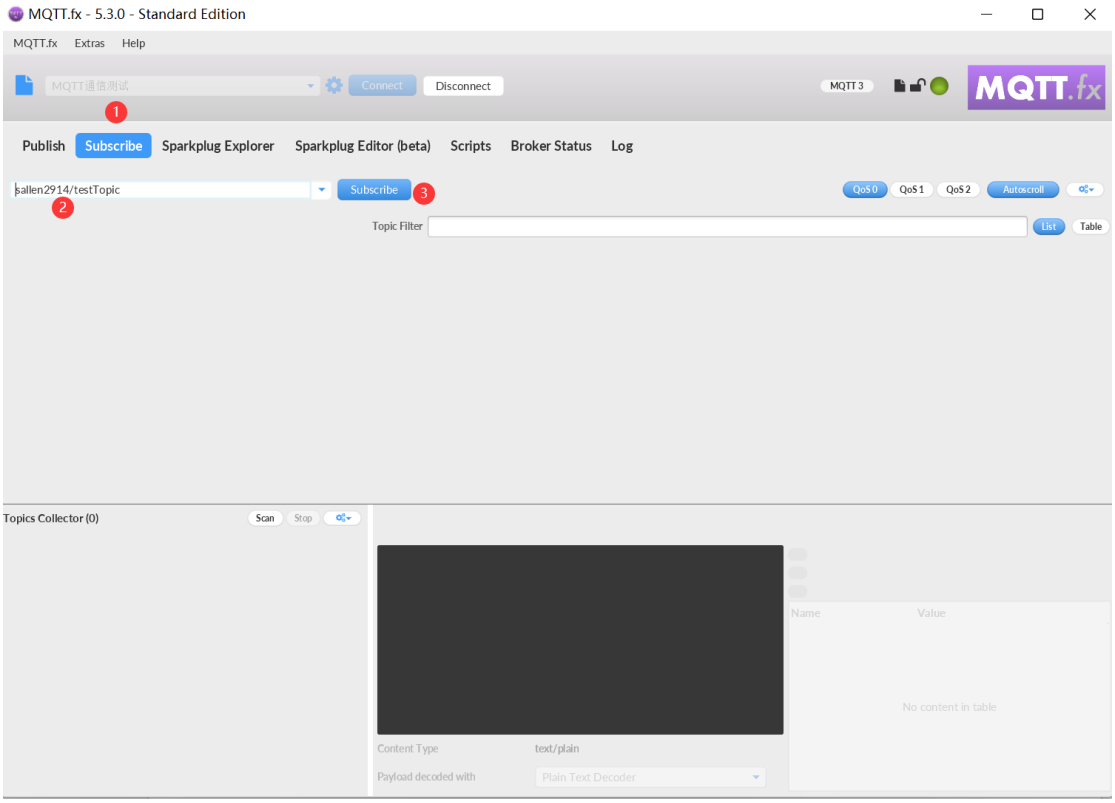
配置好之后点击 connect：

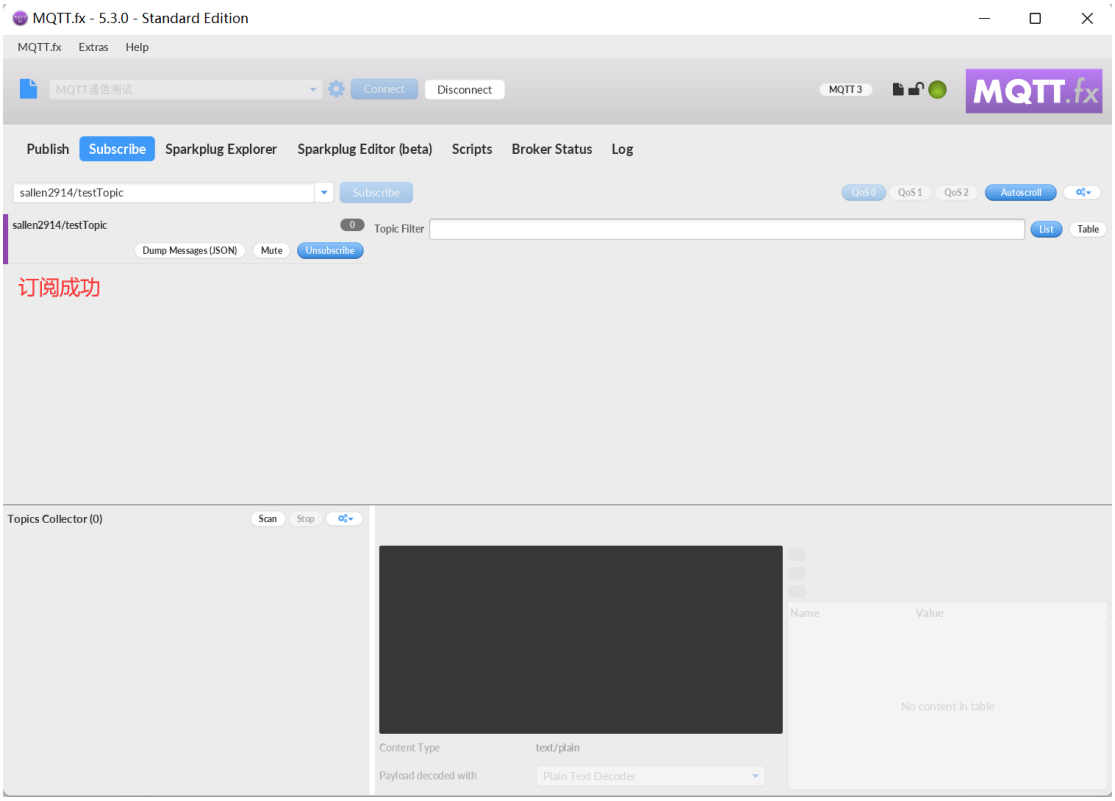


连接成功之后：

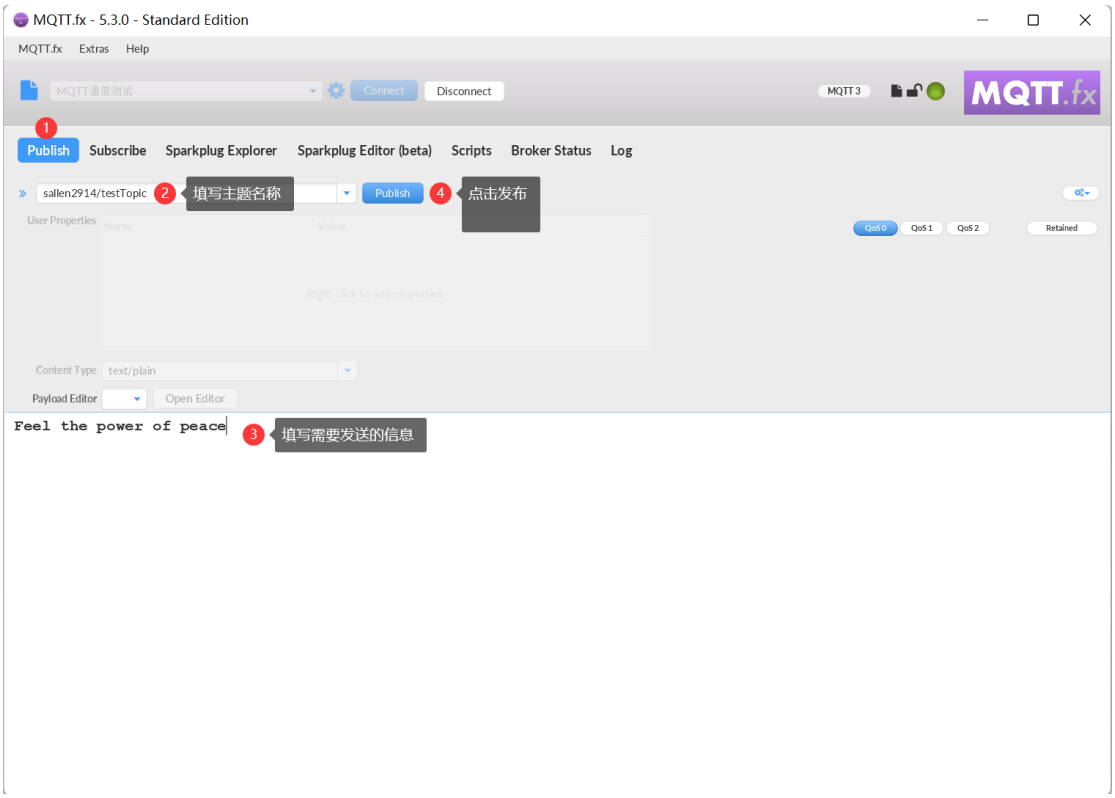

MQTT.fx - 5.3.0 - Standard Edition
MQTT.fx Extras Help
Connect
Disconnect
MQTT 3
MQTT.fx
Publish Subscribe Sparkplug Explorer Sparkplug Editor (beta) Scripts Broker Status Log
Publish
User Properties
Name
Value
QoS 0 QoS 1 QoS 2
Retained
Content Type text/plain
Payload Editor
Open Editor

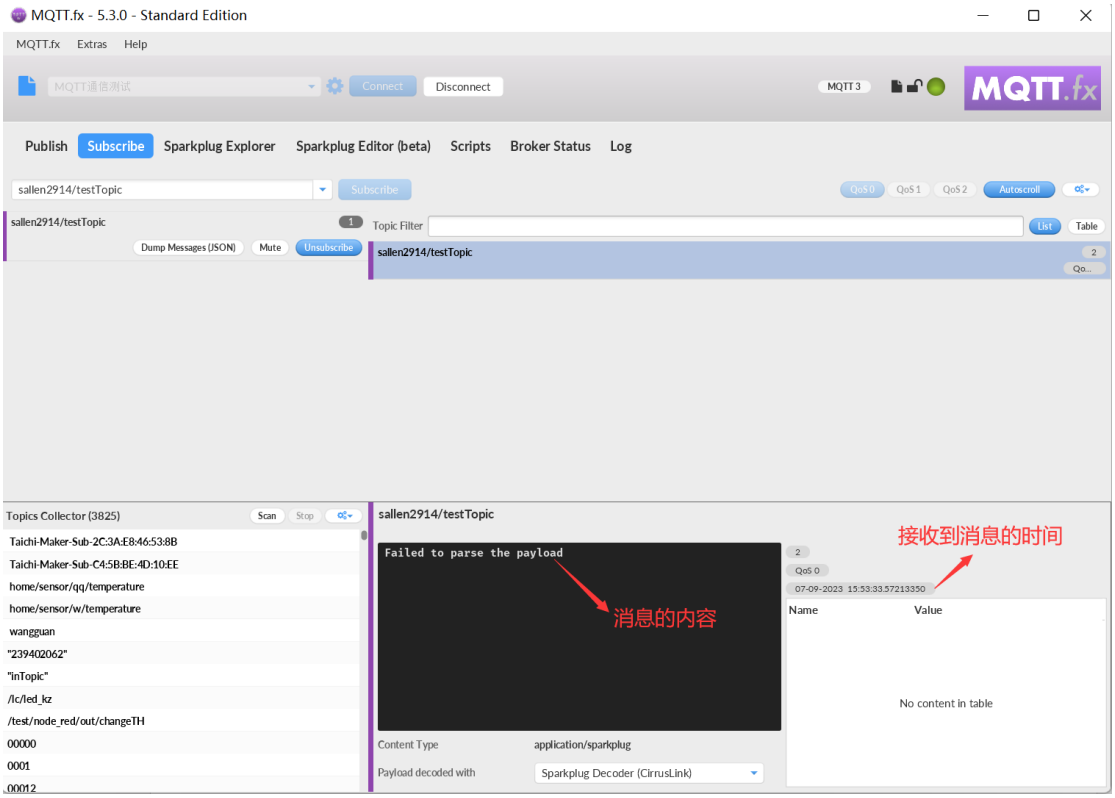
订阅主题：


MQTT.fx - 5.3.0 - Standard Edition
MQTT.fx Extras Help
Connect
Disconnect
MQTT 3
MQTT.fx
1
Publish Subscribe Sparkplug Explorer Sparkplug Editor (beta) Scripts Broker Status Log
sallen2914/testTopic
2
Subscribe
3
QoS 0 QoS 1 QoS 2 Autoscroll
Topic Filter
List Table
Topics Collector (0)
Scan Stop
Name Value
No content in table
Content Type text/plain
Payload decoded with Plain Text Decoder


MQTT.fx - 5.3.0 - Standard Edition
MQTT.fx Extras Help
Connect
Disconnect
MQTT 3
Publish
Subscribe
Sparkplug Explorer
Sparkplug Editor (beta)
Scripts
Broker Status
Log
sallen2914/testTopic
Subscribe
QoS 0
QoS 1
QoS 2
Autoscroll
Topic Filter
List
Table
Dump Messages (JSON)
Mute
Unsubscribe
订阅成功
Topics Collector (0)
Scan
Stop
Name
Value
No content in table
Content Type
text/plain
Payload decoded with
Plain Text Decoder

发布消息：

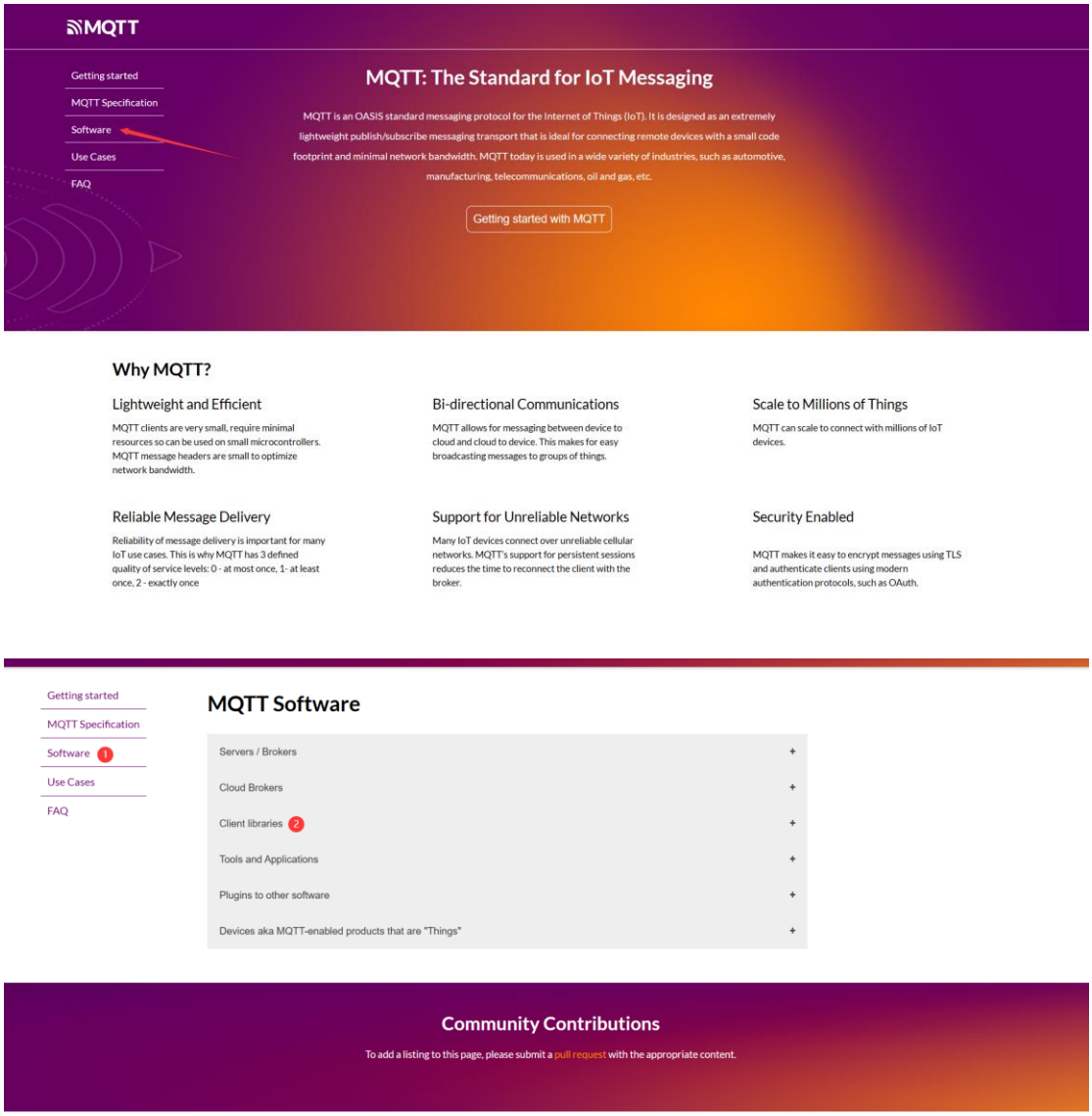

MQTT.fx - 5.3.0 - Standard Edition
MQTT.fx Extras Help
Connect
Disconnect
MQTT 3
1
Publish
Subscribe
Sparkplug Explorer
Sparkplug Editor (beta)
Scripts
Broker Status
Log
sallen2914/testTopic
2
填写主题名称
Publish
4
点击发布
User Properties
Name
Value
QoS 0
QoS 1
QoS 2
Retained
Right click to add properties.
Content Type
text/plain
Payload Editor
Open Editor
Feel the power of peace
3
填写需要发送的信息



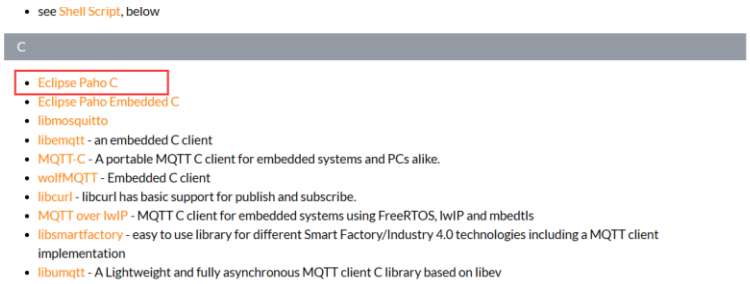
这样可能体现不出效果，因为上例中是自己订阅了“sallen2914/testTopic”主题，接着又是自己向该主题发布消息，虽然是自发自收，但是这个消息肯定是经过了 MQTT 服务端的。既然如此，大家可以找来另一台电脑，在另一台电脑上也安装 MQTT.fx 客户端软件，一台电脑作为主题订阅者、而另一台电脑作为消息发布者进行测试。除此之外，还有一个更简单的办法，直接在电脑上运行两个 MQTT.fx 进程，连接服务器时使用不同的 clientId，这样就是两个不同的 MQTT 客户端了。

4.移植 MQTT 客户端

前面的示例中，我们使用 MQTT.fx 客户端软件在自己的电脑上进行了测试（或在手机上使用 MQTTool 工具进行测试），如果需要在开发板上进行测试，将开发板作为 MQTT 客户端，我们需要自己去编写客户端程序。首先在编写客户端程序之前，需要移植 MQTT 客户端库到我们的开发板上，基于 MQTT 客户端库来编写一个 MQTT 客户端应用程序。那么接下来笔者将向大家介绍如何移植 MQTT 客户端库。如何下载 MQTT 客户端库源码包？首先我们进入到 MQTT 的官网地址:https://mqtt.org/,点击“Software”链接地址,找到“Client libraries”项，如下所示:

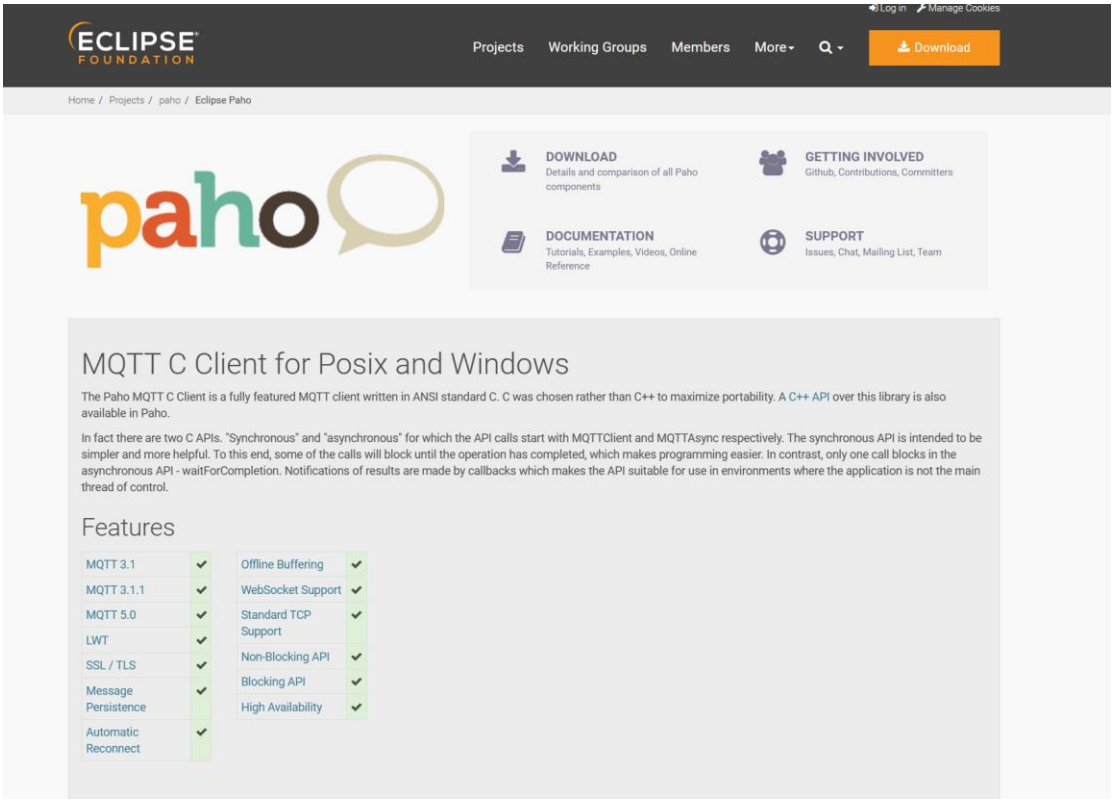


MQTT 客户端库支持多种不同的编程语言，譬如 C、C++、Go、Java、Lua、Objective-C、Python 等，对于我们来说，我们使用的是 C 语言开发，所以要选择 MQTT C 客户端库，如下所示：

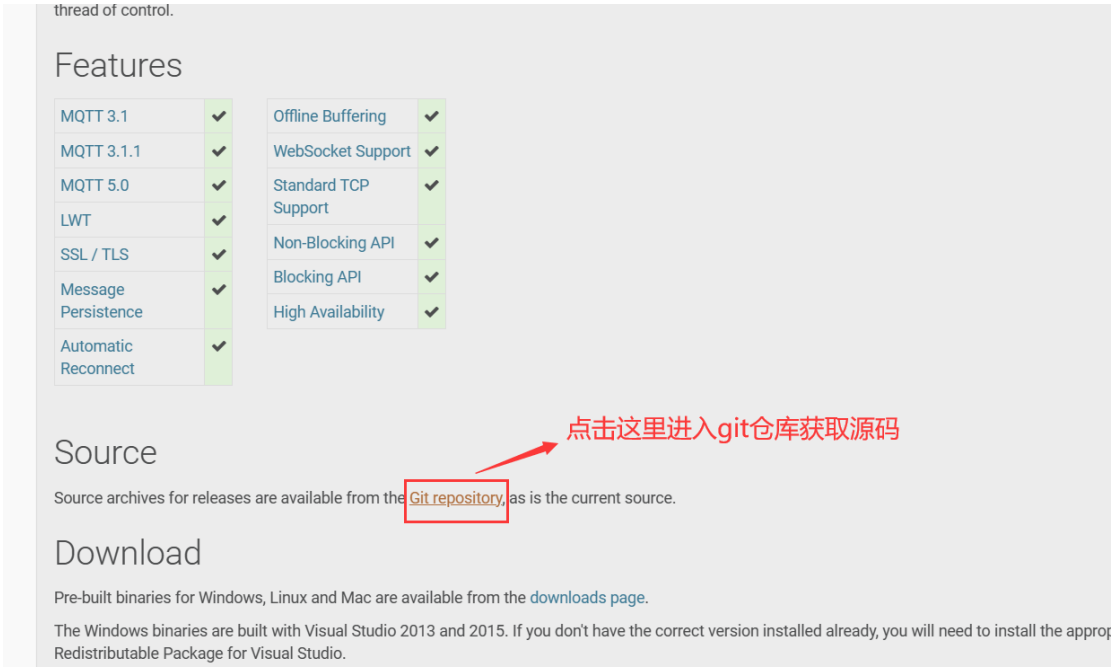


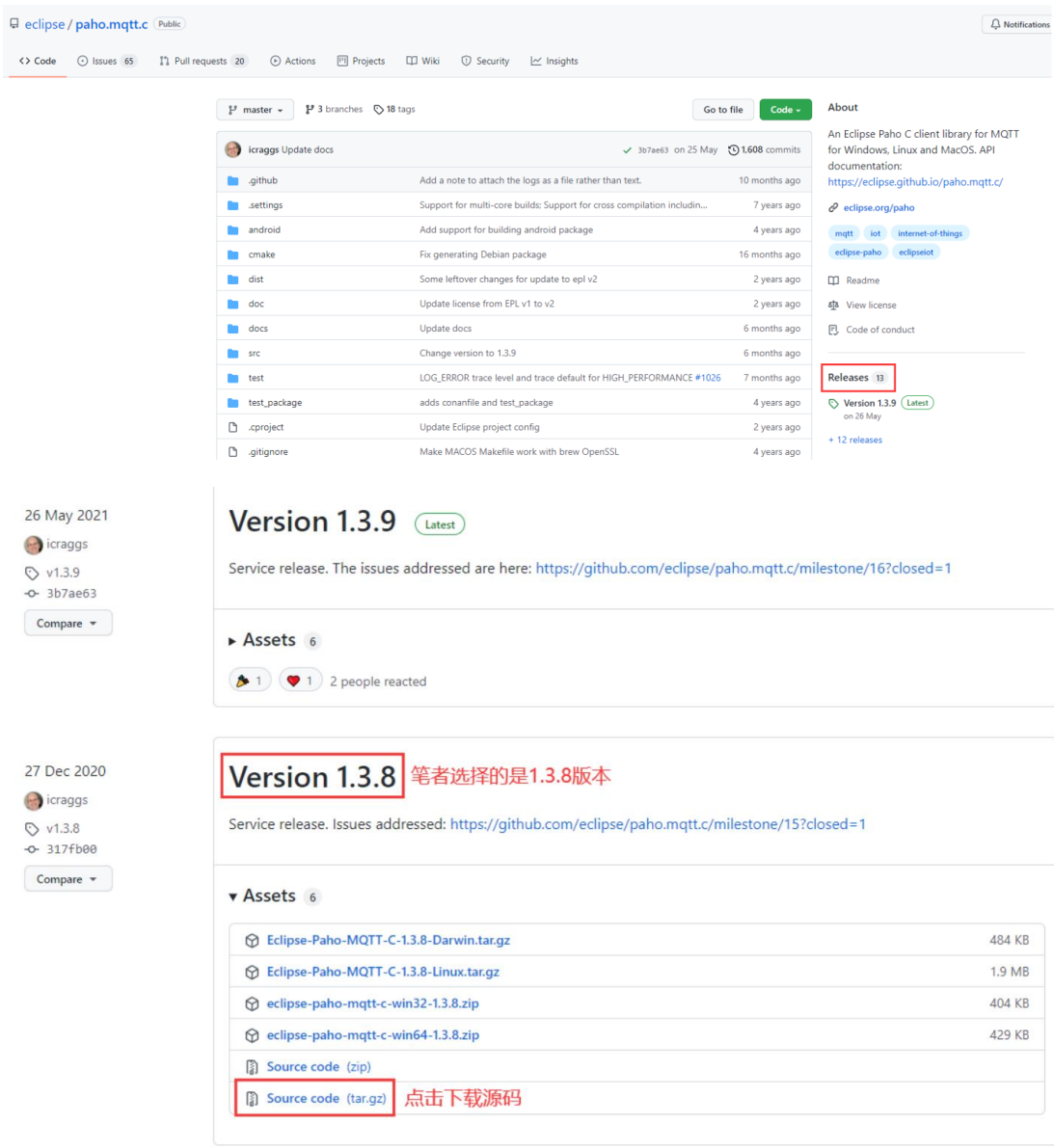
这里有多种不同的 MQTT C 客户端库，笔者推荐大家使用第一个 Eclipse Paho C，这是一个“MQTT C Client for Posix and Windows”，Paho MQTT C 客户端库是用 ANSI 标准 C 编写的功能齐全的 MQTT 客户端库，可运行在 Linux 系统下，支持 MQTT3.1、

MQTT3.1.1、MQTT5.0。点击“Eclipse Paho C”链接地址，如下：



在这个页面中会有一些简单地介绍信息，大家可以自己看一看。我们往下看，找到它的下载地址：





目前最新的版本是 1.3.9，我们不使用最新版本，建议大家使用 1.3.8 版本，如上图所示，点击“Sourcecode (tar.gz)”链接地址下载客户端库源码。

下载成功之后会得到如下压缩文件：

交叉编译 MQTT C 客户端库源码：

将 paho.mqtt.c-1.3.8.tar.gz 压缩文件拷贝到 Ubuntu 系统某个目录下，如下所示：

接着将其解压到当前目录，如下所示：

```
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8$ ls
about.html     cbuild.bat          deploy_rsa.enc  epl-v20       PULL_REQUEST_TEMPLATE.md  travis-build.sh          version.minor
android        cmake               dist            install       README.md                 travis-deploy.sh         version.patch
appveyor.yml   CMakeLists.txt      doc             LICENSE       src                       travis-install.sh
build          CODE_OF_CONDUCT.md  docs            Makefile      test                      travis-setup-deploy.sh
build.xml      CONTRIBUTING.md     edl-v10         notice.html   test_package              version.major
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8$
```

解压成功之后会得到 paho.mqtt.c-1.3.8 文件夹，这就是 paho MQTT C 客户端库源码工程，进入到该目录下，可以看到工程顶级目录下有一个 CMakeLists.txt 文件，所以可知这是一个由 cmake 构建的工程。

首先我们要新建一个交叉编译配置文件 arm-linux-setup.cmake，进入到 cmake 目录下，新建 arm-linux-setup.cmake 文件，并输入以下内容：

```
################################
# 配置 ARM 交叉编译
################################
set(CMAKE_SYSTEM_NAME Linux) #设置目标系统名字
set(CMAKE_SYSTEM_PROCESSOR arm) #设置目标处理器架构
# 指定编译器的 sysroot 路径
set(TOOLCHAIN_DIR /opt/st/stm32mp1/3.1-snapshot/sysroots)
set(CMAKE_SYSROOT ${TOOLCHAIN_DIR}/cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi)
# 指定交叉编译器 arm-linux-gcc
set(CMAKE_C_COMPILER ${TOOLCHAIN_DIR}/x86_64-ostl_sdk-linux/usr/bin/arm-ostl-linux-gnueabi/arm-ostl-linux-gnueabi-gcc)
# 为编译器添加编译选项
set(CMAKE_C_FLAGS "-mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7")
set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
################################
# end
################################
~
~
~
~
~
```

这是配置交叉编译，需要根据自己实际情况修改。

编写完成之后保存退出。

回到工程的顶层目录，新建一个名为 build 的目录，如下所示：

```
sallen@sallen-virtualmachine: ~/linux/MQTTtools/paho.mqtt.c-1.3.8
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8$
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8$
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8$
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8$
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8$
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8$ mkdir bulid
```

进入 bulid 目录下执行 cmake 进行构建：

```
/home/sallen/linux/MQTTtools/cmake-3.16.0-Linux-x86_64/bin/cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/install -DCMAKE_TOOLCHAIN_FILE=/home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/cmake -DPAHO_WITH_SSL=TRUE -DPAHO_BUILD_SAMPLES=TRUE ..
```

```
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8/build$
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8/build$ /home/sallen/linux/MQTTtools/cmake-3.16.0-Linux-x86_64/bin/cmake   -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/install  -DCMAKE_TOOLCHAIN_FILE=/home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/cmake -DPAHO_WITH_SSL=TRUE -DPAHO_BUILD_SAMPLES=TRUE ..
-- CMake version: 3.16.0
-- CMake system name: Linux
-- Timestamp is 2023-09-07T08:19:17Z
-- Configuring done
-- Generating done
CMake Warning:
  Manually-specified variables were not used by the project:

    CMAKE_TOOLCHAIN_FILE


-- Build files have been written to: /home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/build
```

/home/sallen/linux/MQTTtools/cmake-3.16.0-Linux-x86_64/bin/cmake 这是第三十二章时笔者下载的 3.16.0 版本的 cmake 工具，您得根据自己的实际路径来指定；CMAKE_BUILD_TYPE 、CMAKE_INSTALL_PREFIX 、CMAKE_TOOLCHAIN_FILE 都是 cmake 变量，CMAKE_INSTALL_PREFIX 这个指定了安装路径，笔者将安装路径设置为顶层目录下的 install 目录。除此之外，还定义了两个缓存变量 PAHO_WITH_SSL 和 PAHO_BUILD_SAMPLES，具体是什么意思大家可以自己查看工程顶级目录下的 README.md 文件，在 README.md 文件中对工程的编译进行了简单介绍。

cmake 执行完毕之后，接着执行 make 编译：

```
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8/build$
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8/build$ make -j6
[  1%] Linking C executable thread
[ 20%] Built target common_ssl_obj
[ 39%] Built target common_obj
Scanning dependencies of target paho-mqtt3cs
Scanning dependencies of target paho-mqtt3c
Scanning dependencies of target paho-mqtt3a
Scanning dependencies of target paho-mqtt3as
[ 40%] Building C object src/CMakeFiles/paho-mqtt3cs.dir/MQTTClient.c.o
[ 40%] Building C object src/CMakeFiles/paho-mqtt3c.dir/MQTTClient.c.o
[ 41%] Building C object src/CMakeFiles/paho-mqtt3a.dir/MQTTAsync.c.o
[ 42%] Building C object src/CMakeFiles/paho-mqtt3as.dir/MQTTAsync.c.o
[ 44%] Built target thread
[ 44%] Linking C shared library libpaho-mqtt3a.so
[ 45%] Linking C shared library libpaho-mqtt3as.so
[ 46%] Built target paho-mqtt3a
[ 46%] Linking C executable MQTTAsync_subscribe
[ 47%] Linking C executable MQTTAsync_publish_time
[ 48%] Linking C executable MQTTAsync_publish
[ 48%] Built target MQTTAsync_publish
[ 50%] Built target MQTTAsync_publish_time
[ 51%] Built target MQTTAsync_subscribe
[ 52%] Linking C executable test45
[ 53%] Linking C executable test4
[ 54%] Linking C executable test8
[ 56%] Built target paho-mqtt3as
[ 57%] Linking C executable test6
[ 58%] Built target test4
[ 58%] Built target test8
[ 59%] Built target test45
[ 60%] Linking C executable test_issue373
[ 61%] Built target test6
[ 61%] Linking C executable test95
[ 62%] Linking C executable test9
[ 63%] Linking C executable test11
[ 64%] Built target test_issue373
[ 65%] Linking C executable paho_c_sub
[ 67%] Built target test95
[ 68%] Built target test9
[ 69%] Linking C shared library libpaho-mqtt3c.so
[ 70%] Built target test11
[ 71%] Linking C executable paho_c_pub
[ 72%] Linking C executable test5
[ 72%] Built target paho-mqtt3c
[ 73%] Built target paho_c_sub
[ 74%] Linking C executable MQTTClient_subscribe
[ 75%] Linking C executable MQTTVersion
[ 76%] Linking C executable MQTTClient_publish
[ 76%] Built target test5
```

这里需要给大家简单地说明一下，事实上，MQTT 客户端库依赖于 openssl 库，所以通常在移植 MQTT 客户端库的时候，需要先移植 openssl、交叉编译 openssl 得到库文件以及头文件，然后再来编译 MQTT 客户端库；但我们这里没有去移植 openssl，原因是，我们的开发板出厂系统中已经移植好了 openssl 库，并且我们所使用的交叉编译器在编译工程源码的时候会链接 openssl 库（sysroot 路径指定的）。编译成功之后，执行 make install 进行安装：

```
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8/build$ make install
[ 19%] Built target common_ssl_obj
[ 38%] Built target common_obj
[ 39%] Built target paho-mqtt3c
[ 41%] Built target paho-mqtt3cs
[ 43%] Built target paho-mqtt3a
[ 47%] Built target paho-mqtt3as
[ 50%] Built target MQTTVersion
[ 52%] Built target paho_cs_sub
[ 55%] Built target paho_cs_pub
[ 57%] Built target paho_c_sub
[ 58%] Built target MQTTAsync_subscribe
[ 59%] Built target MQTTClient_subscribe
[ 62%] Built target paho_c_pub
[ 64%] Built target MQTTClient_publish
[ 65%] Built target MQTTAsync_publish
[ 68%] Built target MQTTAsync_publish_time
[ 70%] Built target MQTTClient_publish_async
[ 72%] Built target test45
[ 73%] Built target test8
[ 75%] Built target test4
[ 76%] Built target test3
[ 77%] Built target test5
[ 78%] Built target test10
[ 79%] Built target test15
[ 81%] Built target test1
[ 84%] Built target test6
[ 86%] Built target test2
[ 88%] Built target test_issue373
[ 90%] Built target test9
[ 91%] Built target test95
[ 93%] Built target test11
[ 94%] Built target test_sync_session_present
[ 96%] Built target test_connect_destroy
[100%] Built target thread
Install the project...
-- Install configuration: "Release"
-- Up-to-date: /home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/install/share/doc/Eclipse Paho C/samples/MQTTAsync_publish.c
-- Up-to-date: /home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/install/share/doc/Eclipse Paho C/samples/MQTTAsync_publish_time.c
-- Up-to-date: /home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/install/share/doc/Eclipse Paho C/samples/MQTTAsync_subscribe.c
-- Up-to-date: /home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/install/share/doc/Eclipse Paho C/samples/MQTTClient_publish.c
-- Up-to-date: /home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/install/share/doc/Eclipse Paho C/samples/MQTTClient_publish_async.c
-- Up-to-date: /home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/install/share/doc/Eclipse Paho C/samples/MQTTClient_subscribe.c
```

对安装目录下的文件夹进行简单介绍：

进入到安装目录下：

```
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8/build$ cd ..
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8$ ls
about.html     cbuild.bat          deploy_rsa.enc  epl-v20      PULL_REQUEST_TEMPLATE.md  travis-build.sh     version.minor
android        cmake               dist            install      README.md                 travis-deploy.sh    version.patch
appveyor.yml   CMakeLists.txt      doc             LICENSE      src                       travis-install.sh
build          CODE_OF_CONDUCT.md  docs            Makefile     test                      travis-setup-deploy.sh
build.xml      CONTRIBUTING.md     edl-v10         notice.html  test_package              version.major
```

在安装目录下有 bin、include、lib 以及 share 这 4 个文件夹，bin 目录下包含了一些简单的测试 demo，lib 目录下包含了我们编译出来的库文件，如下所示：

```
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8/install/lib$ ls
cmake              libpaho-mqtt3a.so.1      libpaho-mqtt3as.so.1      libpaho-mqtt3c.so.1      libpaho-mqtt3cs.so.1
libmqtt.tar.gz     libpaho-mqtt3a.so.1.3.8  libpaho-mqtt3as.so.1.3.8  libpaho-mqtt3c.so.1.3.8  libpaho-mqtt3cs.so.1.3.8
libpaho-mqtt3a.so  libpaho-mqtt3as.so       libpaho-mqtt3c.so         libpaho-mqtt3cs.so
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8/install/lib$
```

一共有 4 种类型的库，这里我们简单地介绍一下：

- libpaho-mqtt3a.so：异步模式 MQTT 客户端库（不支持 SSL）。
- libpaho-mqtt3as.so：异步模式 MQTT 客户端库（支持 SSL）。
- libpaho-mqtt3c.so：同步模式 MQTT 客户端库（不支持 SSL）。
- libpaho-mqtt3cs.so：支持 SSL 的同步模式客户端库（支持 SSL）

Paho MQTT C 客户端库支持同步操作模式和异步操作模式两种，关于它们之间的区别笔者不做介绍，顶级目录下 docs/MQTTClient/html/async.html 文档（直接双击打开）中对此有相应的解释，有兴趣的可以看一看；docs 目录下提供了很多供用户参考的文档，包括 API 使用说明、示例代码等等，在后续的学习过程中，可以查看这些文档获取帮助。

MQTT 中使用 SSL/TLS 来提供安全性（由 openssl 提供），使用 SSL 来做一些加密验

证,使得数据传输更加安全可靠。以上便给大家简单地介绍了下这 4 种库文件之间的区别，那后续我们将使用 libpaho-mqtt3c.so。

介绍完库文件之后，再来看看头文件，进入到 include 目录下：

```
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8/install/include$ ls
MQTTAsync.h   MQTTClientPersistence.h   MQTTProperties.h   MQTTSubscribeOpts.h
MQTTClient.h  MQTTExportDeclarations.h  MQTTReasonCodes.h
```

在我们的 MQTT 客户端应用程序中只需要包含 MQTTAsync.h 或 MQTTClient.h 头文件即可，其它那些头文件会被这两个头文件所包含；MQTTAsync.h 是异步模式客户端库对外的头文件，而 MQTTClient.h 则是同步模式客户端库对外的头文件。因为后续我们将使用同步模式，所以到时在我们的应用程序中需要包含 MQTTClient.h 头文件。

拷贝库文件到开发板：

使用 tar -czf libmqtt.tar.gz ./*命令打包库文件：

```
sallen@sallen-virtualmachine:~/linux/MQTTtools/paho.mqtt.c-1.3.8/install/lib$ ls
cmake                libpaho-mqtt3a.so.1      libpaho-mqtt3as.so.1      libpaho-mqtt3c.so.1      libpaho-mqtt3cs.so.1
libmqtt.tar.gz       libpaho-mqtt3a.so.1.3.8  libpaho-mqtt3as.so.1.3.8  libpaho-mqtt3c.so.1.3.8  libpaho-mqtt3cs.so.1.3.8
libpaho-mqtt3a.so    libpaho-mqtt3as.so       libpaho-mqtt3c.so         libpaho-mqtt3cs.so
```

使用 scp 命令将压缩包文件 libmqtt.tar.gz 拷贝到开发板 Linux 系统/home/root 目录下：

scp libmqtt.tar.gz@192.168.7.1:/home/root

然后将其解压到开发板的/usr/lib 目录：

tar -xzf libmqtt.tar.gz -C /usr/lib

5. 编写客户端程序

arm-linux-setup.cmake 文件

arm-linux-setup.cmake 源文件用于配置 cmake 交叉编译，其内容如下所示：

```
#################################
# 配置ARM交叉编译
#################################
set(CMAKE_SYSTEM_NAME Linux)      #设置目标系统名字
set(CMAKE_SYSTEM_PROCESSOR arm) #设置目标处理器架构

# 指定编译器的sysroot路径
set(TOOLCHAIN_DIR /opt/st/stm32mp1/3.1-snapshot/sysroots)
set(CMAKE_SYSROOT ${TOOLCHAIN_DIR}/cortexa7t2hf-neon-vfpv4-ostl-linux-gnueabi)

# 指定交叉编译器arm-linux-gcc
set(CMAKE_C_COMPILER ${TOOLCHAIN_DIR}/x86_64-ostl_sdk-linux/usr/bin/arm-ostl-linux-gnueabi/arm-ostl-linux-gnueabi-gcc)

# 为编译器添加编译选项
set(CMAKE_C_FLAGS "-mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7")

set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
#################################
# end
#################################
```

CMakeLists.txt 文件

```
#*****************************************************************************
#  Copyright © ALIENTEK Co., Ltd. 1998-2021. All rights reserved.
#
#  顶层CMakeLists.txt
#  All rights reserved. This program and the accompanying materials
#  are made available under the terms of the Eclipse Public License v2.0
#  and Eclipse Distribution License v1.0 which accompany this distribution.
#*****************************************************************************/
cmake_minimum_required(VERSION 2.8.12)
project(MQTTClient C)
message(STATUS "CMake version: " ${CMAKE_VERSION})
message(STATUS "CMake system name: " ${CMAKE_SYSTEM_NAME})
message(STATUS "CMake system processor: " ${CMAKE_SYSTEM_PROCESSOR})

# 设置可执行文件输出路径
set(EXECUTABLE_OUTPUT_PATH ${PROJECT_BINARY_DIR}/bin)

# 定义可执行文件目标
add_executable(mqttClient mqttClient.c)

# 指定MQTT客户端库头文件路径、库路径以及链接库
# ***大家需要根据MQTT的实际安装路径设置***
target_include_directories(mqttClient PRIVATE /home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/install/include)#MQTT头文件搜索路径
target_link_directories(mqttClient PRIVATE /home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/install/lib)#MQTT库文件搜索路径
target_link_libraries(mqttClient PRIVATE paho-mqtt3c)#MQTT链接库 libpaho-mqtt3c.so
```

客户端应用程序源文件 mqttClient.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "MQTTClient.h"    //包含MQTT客户端库头文件

/* ##########################宏定义###################### */
#define BROKER_ADDRESS "tcp://iot.ranye-iot.net:1883"//然也物联平台社区版MQTT服务器地址

/* 客户端id、用户名、密码 *
 * 当您成功申请到然也物联平台的社区版MQTT服务后
 * 然也物联工作人员会给你发送8组用于连接社区版MQTT服务器
 * 的客户端连接认证信息：也就是客户端id、用户名和密码
 * 注意一共有8组，您选择其中一组覆盖下面的示例值
 * 后续我们使用MQTT.fx或MQTTool的时候 也需要使用一组连接认证信息
 * 去连接社区版MQTT服务器！
 * 由于这是属于个人隐私 笔者不可能将自己的信息写到下面 */
#define CLIENTID        "your clientID"     //客户端id
#define USERNAME        "your username"     //用户名
#define PASSWORD        "your code"         //密码

/* 然也物联社区版MQTT服务为每个申请成功的用户
 * 提供了个人专属主题级别，在官方发给您的微信信息中
 * 提到了
 * 以下sallen_mqtt/led_mqtt/ 便是笔者的个人主题级别
 * sallen_mqtt/led_mqtt其实就是笔者申请社区版MQTT服务时注册的用户名
 * 大家也是一样，所以你们需要替换下面的dt_mqtt前缀
 * 换成你们的个人专属主题级别（也就是您申请时的用户名）
 */
#define WILL_TOPIC      "sallen_mqtt/led_mqtt/will"     //遗嘱主题
#define LED_TOPIC       "sallen_mqtt/led"       //LED主题
#define TEMP_TOPIC      "sallen_mqtt/temperature"       //温度主题
/* ################################################## */

static int msgarrvd(void *context, char *topicName, int topicLen,
                    MQTTClient_message *message)
{
    if (!strcmp(topicName, LED_TOPIC)) {//校验消息的主题
        if (!strcmp("2", message->payload))//如果接收到的消息是"2"则设置LED为呼吸灯模式
            system("echo heartbeat > /sys/class/leds/user-led/trigger");
        if (!strcmp("1", message->payload)) {//如果是"1"则LED常量
            system("echo none > /sys/class/leds/user-led/trigger");
            system("echo 1 > /sys/class/leds/user-led/brightness");
        }
        else if (!strcmp("0", message->payload)) {//如果是"0"则LED熄灭
            system("echo none > /sys/class/leds/user-led/trigger");
            system("echo 0 > /sys/class/leds/user-led/brightness");
        }

        // 接收到其它数据 不做处理
    }
```

```
    /* 释放占用的内存空间 */
    MQTTClient_freeMessage(&message);
    MQTTClient_free(topicName);

    /* 退出 */
    return 1;
}

static void connlost(void *context, char *cause)
{
    printf("\nConnection lost\n");
    printf("     cause: %s\n", cause);
}

int main(int argc, char *argv[])
{
    MQTTClient client;
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_willOptions will_opts = MQTTClient_willOptions_initializer;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    int rc;

    /* 创建mqtt客户端对象 */
    if (MQTTCLIENT_SUCCESS !=
            (rc = MQTTClient_create(&client, BROKER_ADDRESS, CLIENTID,
            MQTTCLIENT_PERSISTENCE_NONE, NULL))) {
        printf("Failed to create client, return code %d\n", rc);
        rc = EXIT_FAILURE;
        goto exit;
    }

    /* 设置回调 */
    if (MQTTCLIENT_SUCCESS !=
            (rc = MQTTClient_setCallbacks(client, NULL, connlost,
            msgarrvd, NULL))) {
        printf("Failed to set callbacks, return code %d\n", rc);
        rc = EXIT_FAILURE;
        goto destroy_exit;
    }
```

```
    /* 连接MQTT服务器 */
    will_opts.topicName = WILL_TOPIC;    //遗嘱主题
    will_opts.message = "Unexpected disconnection";//遗嘱消息
    will_opts.retained = 1;//保留消息
    will_opts.qos = 0;        //QoS0

    conn_opts.will = &will_opts;
    conn_opts.keepAliveInterval = 30;    //心跳包间隔时间
    conn_opts.cleansession = 0;        //cleanSession标志
    conn_opts.username = USERNAME;        //用户名
    conn_opts.password = PASSWORD;        //密码
    if (MQTTCLIENT_SUCCESS !=
            (rc = MQTTClient_connect(client, &conn_opts))) {
        printf("Failed to connect, return code %d\n", rc);
        rc = EXIT_FAILURE;
        goto destroy_exit;
    }

    printf("MQTT服务器连接成功!\n");

    /* 发布上线消息 */
    pubmsg.payload = "Online";    //消息的内容
    pubmsg.payloadlen = 6;        //内容的长度
    pubmsg.qos = 0;            //QoS等级
    pubmsg.retained = 1;        //保留消息
    if (MQTTCLIENT_SUCCESS !=
            (rc = MQTTClient_publishMessage(client, WILL_TOPIC, &pubmsg, NULL))) {
        printf("Failed to publish message, return code %d\n", rc);
        rc = EXIT_FAILURE;
        goto disconnect_exit;
    }

    /* 订阅主题 dt_mqtt/led */
    if (MQTTCLIENT_SUCCESS !=
            (rc = MQTTClient_subscribe(client, LED_TOPIC, 0))) {
        printf("Failed to subscribe, return code %d\n", rc);
        rc = EXIT_FAILURE;
        goto disconnect_exit;
    }
```

```
    /* 向服务端发布芯片温度信息 */
    for ( ;; ) {

        MQTTClient_message tempmsg = MQTTClient_message_initializer;
        char temp_str[10] = {0};
        int fd;

        /* 读取温度值 */
        fd = open("/sys/class/hwmon/hwmon0/temp1_input", O_RDONLY);
        read(fd, temp_str, sizeof(temp_str));//读取temp属性文件即可获取温度
        close(fd);

        /* 发布温度信息 */
        tempmsg.payload = temp_str;//消息的内容
        tempmsg.payloadlen = strlen(temp_str);        //内容的长度
        tempmsg.qos = 0;                    //QoS等级
        tempmsg.retained = 1;               //保留消息
        if (MQTTCLIENT_SUCCESS !=
            (rc = MQTTClient_publishMessage(client, TEMP_TOPIC, &tempmsg, NULL))) {
            printf("Failed to publish message, return code %d\n", rc);
            rc = EXIT_FAILURE;
            goto unsubscribe_exit;
        }

        sleep(30);          //每隔30秒 更新一次数据
    }

unsubscribe_exit:
    if (MQTTCLIENT_SUCCESS !=
        (rc = MQTTClient_unsubscribe(client, LED_TOPIC))) {
        printf("Failed to unsubscribe, return code %d\n", rc);
        rc = EXIT_FAILURE;
    }
disconnect_exit:
    if (MQTTCLIENT_SUCCESS !=
        (rc = MQTTClient_disconnect(client, 10000))) {
        printf("Failed to disconnect, return code %d\n", rc);
        rc = EXIT_FAILURE;
    }
destroy_exit:
    MQTTClient_destroy(&client);
exit:
    return rc;
}
```

WILL_TOPIC：这是客户端的遗嘱主题。

LED_TOPIC:LED 主题，我们的开发板客户端订阅了该主题，而我们会通过其它客户端，譬如手机或电脑去向这个主题发布信息，那么接收到信息之后根据信息的内容，来对 LED 做出相应的控制，譬如点亮 LED、熄灭 LED。

TEMP_TOPIC：温度主题，我们的开发板客户端会向这个主题发布消息，这个消息的内容就是开发板这个芯片温度值，开发板的温度值怎么获取？对于 MP157 开发板来说，就是读取/sys/class/hwmon/hwmon0/temp1_input 属性文件。同样，其它客户端（譬如手机或电脑）会订阅这个温度主题，所以，手机或电脑就会收到开发板的温度信息。程序中是设置没个 30 秒发一次。

构建、编译

我们直接进行编译，进入到工程目录下的 build 目录中，执行 cmake 构建：

/home/sallen/linux/MQTTtools/cmake-3.16.0-Linux-x86_64/bin/cmake

-DCMAKE_TOOLCHAIN_FILE=/home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/cmake

/arm-linux-setup.cmake -DCMAKE_BUILD_TYPE=Release ..

```
sallen@sallen-virtualmachine:~/vscode_ws/mqtt_prj/build$ /home/sallen/linux/MQTTtools/cmake-3.16.0-Linux-x86_64/bin/cmake  -DCMAKE_T
OOLCHAIN_FILE=/home/sallen/linux/MQTTtools/paho.mqtt.c-1.3.8/cmake/arm-linux-setup.cmake -DCMAKE_BUILD_TYPE=Release ..
-- CMake version: 3.16.0
-- CMake system name: Linux
-- CMake system processor: arm
-- Configuring done
-- Generating done
CMake Warning:
  Manually-specified variables were not used by the project:

    CMAKE_TOOLCHAIN_FILE


-- Build files have been written to: /home/sallen/vscode_ws/mqtt_prj/build
```

执行 make 编译:

```
sallen@sallen-virtualmachine:~/vscode_ws/mqtt_prj/build$ make
Scanning dependencies of target mqttClient
[ 50%] Building C object CMakeFiles/mqttClient.dir/mqttClient.c.o
[100%] Linking C executable bin/mqttClient
[100%] Built target mqttClient
```

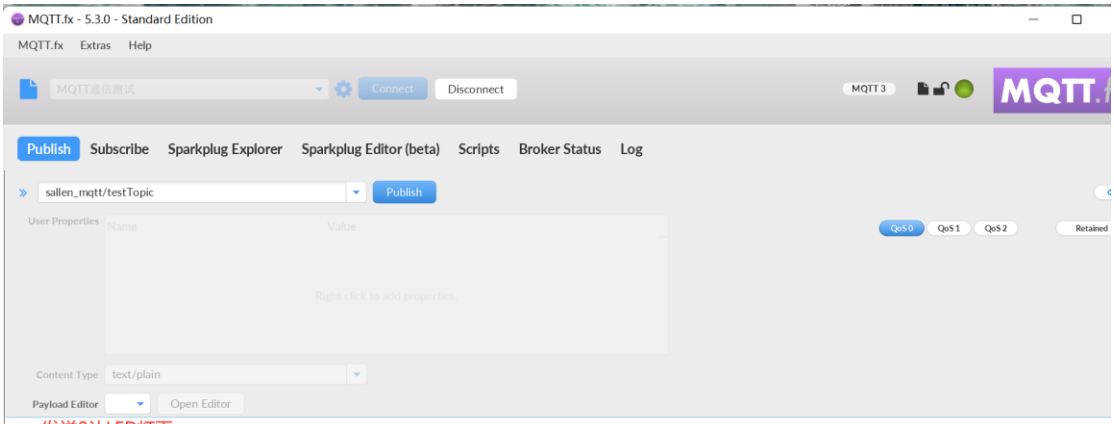
编译成功之后，在 build/bin 目录下生成了可执行文件 mqttClient。

```
sallen@sallen-virtualmachine:~/vscode_ws/mqtt_prj/build/bin$ ls
mqttClient
```

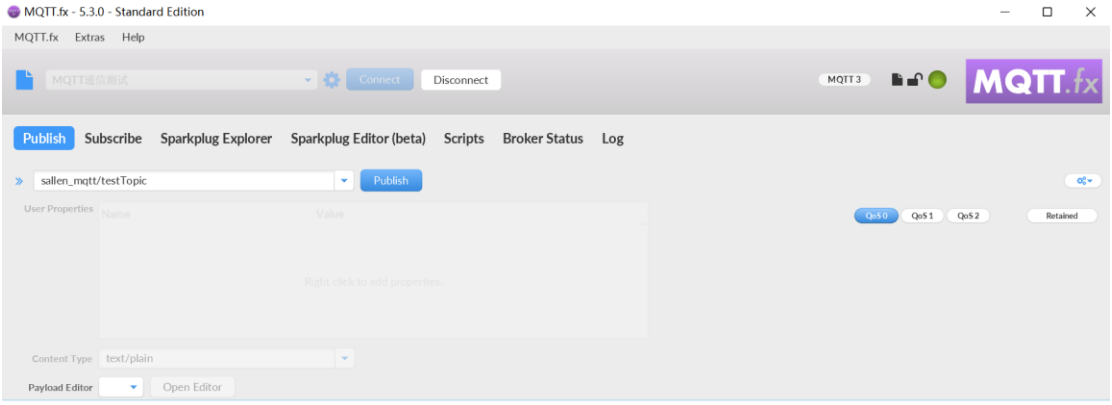
利用 scp 命令将可执行文件 mqttClient 拷贝到开发板 Linux 系统/home/root 目录下。

```
root@ATK-MP157:~# ls
mqttClient  shell
root@ATK-MP157:~#
root@ATK-MP157:~#
root@ATK-MP157:~# ./mqttClient
MQTT服务器连接成功!
```

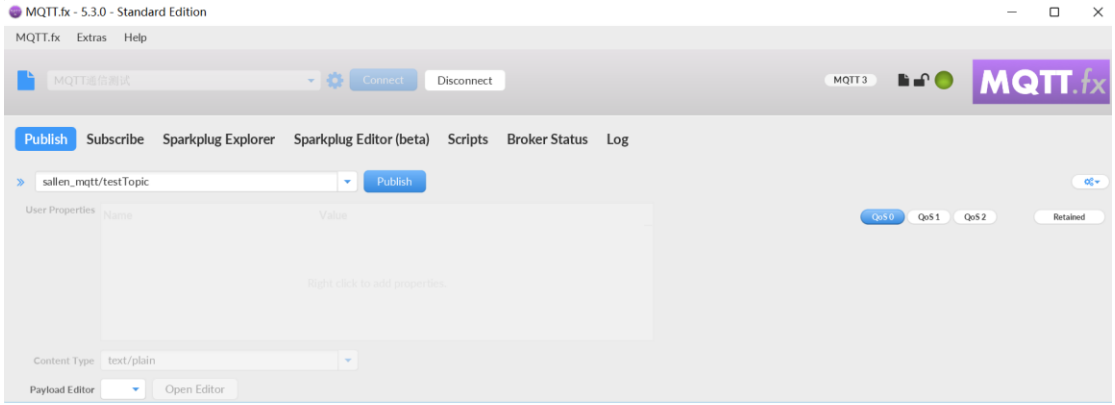
6.演示



0 发送0让LED灯灭



1 发送1让LED灯亮



2 发送2让LED灯处于呼吸灯的模式