交叉编译Qt库

1、环境准备

X86宿主机版本：Ubuntu 22.04.03 LTS (Desktop)

> Ubuntu 22.04宿主机建议配置至少4个cpu核，内存要4G以上

ARM目标系统版本: Orange Pi 3.0.6 Jammy （aarch64）

2 、编译环境搭建（Ubuntu 22.04）

2.1 、编译指定版本的QT库

编译目标版本： Qt 5.15.10

2.2. 创建编译目录并下载源码

```
mkdir qt5/build -p
mkdir qt5/sysroot
wget  https://mirrors.ustc.edu.cn/qtproject/archive/qt/5.15/5.15.10/single/qt-everywhere-opensource-src-5.15.10.tar.xz   #下载qt5.15.10源码
tar -xvf qt-everywhere-opensource-src-5.15.10.tar.xz  -C qt5/
```

2.3. 在宿主机上，使用下面命令安装相关工具或者库:

```
sudo apt update
sudo apt install gcc git bison  gperf pkg-config gdb-multiarch
sudo apt install build-essential
sudo apt install g++-aarch64-linux-gnu gcc-aarch64-linux-gnu
```

2.4. 搭建sysroot依赖环境

由于交叉编译的qt程序最终是要运行在arm开发板上的，在软件编译过程中，有时需要指定路径的头文件和链接指定路径的库。但在交叉编译时，需要在本地系统（X86宿主机）进行编译，而编译出的软件在目标系统（ARM）运行，这就存在编译时指定的路径和运行时的路径不一致的矛盾。因此在编译qt的时候，需要提供和开发板一样的库依赖和头文件进行编译， 因此需要在qt5源码编译时，配置--sysroot=dir的情况参数，dir就被作为逻辑根目录，链接器将在dir/usr/lib中搜索库文件。此处的dir（逻辑根目录）可以通过以下几种方式获得：

a、直接从ARM目标上获取sysroot环境

首先，在ARM开发板上安装如下依赖:

```
# 更新源
sudo apt update

# 安装x11相关库
sudo apt install -y libx11-dev freetds-dev libxcb-xinput-dev libpq-dev libiodbc2-dev firebird-dev
sudo apt install -y  libxcb-image0-dev libxcb-shm0 libxcb-shm0-dev libxcb-icccm4 libxcb-icccm4-dev libxcb-sync1
sudo apt install -y libxcb-sync-dev libxcb-render-util0 libxcb-render-util0-dev libxcb-xfixes0-dev libxrender-dev
```

```
sudo apt install -y  libxcb-shape0-dev libxcb-randr0-dev libxcb-glx0-dev libxi-dev libdrm-dev
sudo apt install -y libxcb-xinerama0-dev libatspi2.0-dev libxcursor-dev libxcomposite-dev libxdamage-dev
sudo apt install -y libxss-dev libxtst-dev  libcap-dev libxrandr-dev libdirectfb-dev libaudio-dev
sudo apt install -y  libxkbcommon-x11-dev libxcb-keysyms1-dev libx11-xcb-dev

# 安装gstreamer相关库
sudo apt install -y libgstreamer1.0-0 gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly gstreamer1.0-libav  gstreamer1.0-tools libunwind-dev

# 安装opengl相关库
sudo apt install -y  libegl1-mesa-dev libgles2-mesa-dev libgles2-mesa

# 安装其他库
sudo apt install -y libfreetype6-dev libicu-dev libsqlite3-dev libasound2-dev libnss3-dev libxss-dev libxtst-dev libcap-dev libxrandr-dev libdirectfb-dev libaudio-dev libavcodec-dev libavformat-dev libswscale-dev libts-dev libfontconfig1-dev libssl-dev
libdbus-1-dev rsyslog  libjpeg-dev
```

接着，将已安装好镜像及依赖库的TF卡接入宿主机，并将其挂载上来， 并将整个根分区数据拷贝出来

```
udisksctl mount -b /dev/sdb1 # /dev/sdb1视实际的设备节点而定， 有可能是sdc1或者其他
cd /media/$(whoami)/TF/ # TF为SD卡视实际的决定，有可能是TF卡的UUID或者名称
sudo cp * ~/qt5/sysroot/ -af
```

有些软连接库是使用绝对路径，交叉编译Qt源码时会出错。我们可以使用symlinks修改绝对路径为相对路径，安装命令：

```
sudo apt-get install symlinks
sudo  symlinks -rc ~/qt5/sysroot
```

b、利用ubuntu-base搭建sysroot环境

宿主机安装qemu工具:

```
sudo apt install qemu-user-static
```

从ubuntu base官网下载ubuntu-base-22.04.1-base-arm64.tar.gz包:

```
wget  https://cdimage.ubuntu.com/ubuntu-base/releases/22.04.1/release/ubuntu-base-22.04.1-base-arm64.tar.gz
```

解压ubuntu-base:

```
tar -xvf ubuntu-base-22.04.1-base-arm64.tar.gz -C qt5/sysroot
```

配置qt5/sysroot/etc/resolve.conf文件:

```
#vim.tiny qt5/sysroot/etc/rlsesolv.conf
nameserver 8.8.8.8
namseserver 114.114.114.114
```

创建用于chroot的ch-mount.sh脚本

```
#!/bin/bash

#vim.tiny ch-mount.sh

mnt() {
    echo "MOUNTING"
    sudo mount -t proc /proc ${2}/proc
    sudo mount -t sysfs /sys ${2}/sys
    sudo mount -o bind /dev ${2}/dev
    sudo mount -o bind /dev/pts ${2}/dev/pts
    sudo mount -t tmpfs -o  mode=0777 tmpfs ${2}/tmp/
    sudo chroot ${2}
}
umnt() {
    echo "UNMOUNTING"
    sudo umount ${2}/proc
    sudo umount ${2}/sys
    sudo umount ${2}/dev/pts
    sudo umount ${2}/dev
    sudo umount ${2}/tmp
}
if [ "$1" == "-m" ] && [ -n "$2" ] ;
then
    mnt $1 $2
elif [ "$1" == "-u" ] && [ -n "$2" ];
then
    umnt $1 $2
else
    echo ""
    echo "Either 1'st, 2'nd or both parameters were missing"
    echo ""
    echo "1'st parameter can be one of these: -m(mount) OR -u(umount)"
    echo "2'nd parameter is the full path of rootfs directory(with trailing '/')"
    echo ""
    echo "For example: ch-mount -m /media/sdcard/"
    echo ""
    echo 1st parameter : ${1}
    echo 2nd parameter : ${2}
fi
```

给ch-mount.sh提供权限

```
sudo chmod +x ch-mount.sh
```

修改 sysroot的默认sources.list

```
# vim.tiny qt5/sysroot/etc/apt/sources.list
```

```
# 阿里云Debian稳定版仓库
deb http://repo.huaweicloud.com/ubuntu-ports/ jammy main restricted universe
multiverse
deb-src http://repo.huaweicloud.com/ubuntu-ports/ jammy main restricted universe
multiverse

# 阿里云更新仓库
deb http://repo.huaweicloud.com/ubuntu-ports/ jammy-updates main restricted
universe multiverse
deb-src http://repo.huaweicloud.com/ubuntu-ports/ jammy-updates main restricted
universe multiverse

# 阿里云安全更新仓库
deb http://repo.huaweicloud.com/ubuntu-ports/ jammy-security main restricted
universe multiverse
deb-src http://repo.huaweicloud.com/ubuntu-ports/ jammy-security main restricted
universe multiverse

# 可选：阿里云的第三方软件仓库（如果需要）
deb http://repo.huaweicloud.com/ubuntu-ports/ jammy-backports main restricted
universe multiverse
deb-src http://repo.huaweicloud.com/ubuntu-ports/ jammy-backports main restricted
universe multiverse
```

切换进去sysroot目录：

```
linux@linux-virtual-machine:~$ sudo ./ch-mount.sh  -m qt5/sysroot
MOUNTING
root@linux-virtual-machine:/#
```

然后根据前面a、直接从ARM目标上获取sysroot环境方式， 安装ARM环境依赖（*注：此时不需要sudo运行，默认就拥有root权限，另外此时和TF卡无关，不需要进行TF卡相关的操作*）

```
# 更新源
sudo apt update

# 安装x11相关库
apt install -y libx11-dev freetds-dev libxcb-xinput-dev libpq-dev libiodbc2-dev
firebird-dev
apt install -y  libxcb-image0-dev libxcb-shm0 libxcb-shm0-dev libxcb-icccm4
libxcb-icccm4-dev libxcb-sync1
apt install -y libxcb-sync-dev libxcb-render-util0 libxcb-render-util0-dev
libxcb-xfixes0-dev libxrender-dev
apt install -y  libxcb-shape0-dev libxcb-randr0-dev libxcb-glx0-dev libxi-dev
libdrm-dev
apt install -y libxcb-xinerama0-dev libatspi2.0-dev libxcursor-dev libxcomposite-
dev libxdamage-dev
apt install -y libxss-dev libxtst-dev  libcap-dev libxrandr-dev libdirectfb-dev
libaudio-dev
apt install -y  libxkbcommon-x11-dev libxcb-keysyms1-dev libx11-xcb-dev

# 安装gstreamer相关库
```

```
apt install -y libgstreamer1.0-0 gstreamer1.0-plugins-base gstreamer1.0-plugins-
good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly gstreamer1.0-libav
gstreamer1.0-tools libunwind-dev

# 安装opengl相关库
apt install -y  libegl1-mesa-dev libgles2-mesa-dev libgles2-mesa

# 安装其他库
apt install -y libfreetype6-dev libicu-dev libsqlite3-dev libasound2-dev libnss3-
dev libxss-dev libxtst-dev libcap-dev libxrandr-dev libdirectfb-dev libaudio-dev
libavcodec-dev libavformat-dev libswscale-dev libts-dev libfontconfig1-dev
libssl-dev
libdbus-1-dev rsyslog  libjpeg-dev
```

另外在sysroot的qemu环境中安装dbus-x11

```
update

root@linux-virtual-machine:/# apt install dbus-x11
```

安装完成后，执行exit退出，并卸载挂载在sysroot上的相关节点,同时重新修改下sysroot目录下的软链：

```
root@linux-virtual-machine:/# exit
exit
linux@linux-virtual-machine:~$ sudo ./ch-mount.sh  -u qt5/sysroot/
UNMOUNTING
```

在根据上面 方案a和方案b，两种方案准备完sysroot后，需要同时重新修改下sysroot目录下里面部分库的软链接，否则有些库是以决定路径的形式链接的，会导致库链接是吧。首先创建即将执行的python脚本：

```
#!/usr/bin/env python3
import sys
import os

# vim.tiny ~/qt5/sysroot.py
# Take a sysroot directory and turn all the abolute symlinks and turn them into
# relative ones such that the sysroot is usable within another system.

if len(sys.argv) != 2:
    print("Usage is " + sys.argv[0] + "<directory>")
    sys.exit(1)

topdir = sys.argv[1]
topdir = os.path.abspath(topdir)

def handlelink(filep, subdir):
    link = os.readlink(filep)
    if link[0] != "/":
        return
    if link.startswith(topdir):
```

```
        return
    #print("Replacing %s with %s for %s" % (link, topdir+link, filep))
    print("Replacing %s with %s for %s" % (link, os.path.relpath(topdir+link,
subdir), filep))
    os.unlink(filep)
    os.symlink(os.path.relpath(topdir+link, subdir), filep)

for subdir, dirs, files in os.walk(topdir):
    for f in files:
        filep = os.path.join(subdir, f)
        if os.path.islink(filep):
            #print("Considering %s" % filep)
            handlelink(filep, subdir)
```

然后执行sysroot.py脚本

```
cd qt5/
sudo  chmod +x sysroot.py
sudo  python3 ./sysroot.py sysroot
```

2.5 交叉编译qt5

创建build-qt5.sh脚本，添加配置qt5编译脚本

```
#!/bin/sh

#vim.tiny ~/qt5/build-qt5.sh

DEVICE=linux-orangepi-g++
SCRIPT_PATH=$(pwd)

#QT版本的前缀和后缀,默认使用5.15.10
QT_VER_PREFIX=5.15
QT_VER_SUFFIX=.10
MAJOR_NAME=qt-everywhere-src

QT_VER=${QT_VER_PREFIX}${QT_VER_SUFFIX}

#源码包解压后的名称
PACKAGE_NAME=${MAJOR_NAME}-${QT_VER_PREFIX}${QT_VER_SUFFIX}

#定义编译后安装--生成的文件,文件夹位置路径
INSTALL_PATH_EXT=/opt/${PACKAGE_NAME}/ext
INSTALL_PATH_HOST=/opt/${PACKAGE_NAME}/host

#定义sysroot目录路径
SYSROOT_PATH=~/qt5/sysroot

#添加交叉编译工具链路径
CROSS_CHAIN_PREFIX=/usr/bin/aarch64-linux-gnu-

CONFIG_PATH=${SCRIPT_PATH}/${PACKAGE_NAME}/qtbase/mkspecs/devices/${DEVICE}

#qmake.conf路径
```

```
CONFIG_FILE=${CONFIG_PATH}/qmake.conf

#创建和修改平台配置信息
do_add_qmake_conf () {
     cd ${PACKAGE_NAME}

     if [ ! -d "${CONFIG_PATH}" ];then
         cp -a ${SCRIPT_PATH}/${PACKAGE_NAME}/qtbase/mkspecs/devices/linux-
generic-g++ ${CONFIG_PATH}
     fi

     echo "#" > ${CONFIG_FILE}
     echo "#" >> ${CONFIG_FILE}
     echo "" >> ${CONFIG_FILE}
     echo "include(../common/linux_device_pre.conf)" >> ${CONFIG_FILE}
     echo "" >> ${CONFIG_FILE}
     echo "QMAKE_CFLAGS            = -march=armv8-a" >> ${CONFIG_FILE}
     echo "QMAKE_CXXFLAGS          = \$\$QMAKE_CFLAGS" >> ${CONFIG_FILE}
     echo "QMAKE_LFLAGS += -static-libstdc++" >> ${CONFIG_FILE}
     echo "" >> ${CONFIG_FILE}

     echo "QMAKE_INCDIR_POST += \
         \$\$[QT_SYSROOT]/usr/include \
         \$\$[QT_SYSROOT]/usr/include/\$\${GCC_MACHINE_DUMP} " >> ${CONFIG_FILE}

     echo "QMAKE_LIBDIR_POST += \
         \$\$[QT_SYSROOT]/usr/lib \
         \$\$[QT_SYSROOT]/lib/\$\${GCC_MACHINE_DUMP} \
         \$\$[QT_SYSROOT]/usr/lib/\$\${GCC_MACHINE_DUMP} " >> ${CONFIG_FILE}

     echo "QMAKE_RPATHLINKDIR_POST += \
         \$\$[QT_SYSROOT]/usr/lib \
         \$\$[QT_SYSROOT]/usr/lib/\$\${GCC_MACHINE_DUMP} \
         \$\$[QT_SYSROOT]/lib/\$\${GCC_MACHINE_DUMP} " >> ${CONFIG_FILE}

     echo "" >> ${CONFIG_FILE}
     echo "DISTRO_OPTS += aarch64" >> ${CONFIG_FILE}
     echo "DISTRO_OPTS += deb-multi-arch" >> ${CONFIG_FILE}
     echo "" >> ${CONFIG_FILE}
     echo "include(../common/linux_arm_device_post.conf)" >> ${CONFIG_FILE}
     echo "load(qt_config)" >> ${CONFIG_FILE}

     cat ${CONFIG_FILE}
}

#配置选项
do_configure () {
      cd ${SCRIPT_PATH}
      mkdir build -p
      cd build

     ../${PACKAGE_NAME}/configure \
     -sysroot ${SYSROOT_PATH}  \
     -hostprefix ${INSTALL_PATH_HOST} \
     -extprefix ${INSTALL_PATH_EXT} \
     -device ${DEVICE} \
```

```
    -device-option CROSS_COMPILE=${CROSS_CHAIN_PREFIX} \
    -release \
    -opensource \
    -confirm-license \
    -nomake tests  \
    -make libs \
    -no-opengl\
    -skip qtscript \
    -skip qtwebengine  \
    -skip qtdeclarative \
    -no-use-gold-linker  \
    -v \
    -recheck-all
}

# 添加配置文件
do_add_qmake_conf

# 配置
do_configure

exit 0
```

增加脚本权限,并执行改脚本

```
cd ~/qt5
sudo chmod +x build-qt5.sh
./build-qt5.sh
```

配置完成后，利用gmake命令进行编译和安装

```
cd ~/qt5/build
gmake -j 6
sudo  gmake install
```

安装完成后，正常会在宿主机/opt目录下出现/opt/qt-everywhere-src-5.15.10/ext和/opt/qt-everywhere-src-5.15.10/ext目录

其中ext目录就是交叉编译出来用于ARM开发版的QT5的相关库及配置文件。 host目录用于交叉编译qt工程的相关程序及配置

2.6. 移植qt库

1. 直接将/opt/qt-everywhere-src-5.15.10/ext 拷贝到arm的/opt目录下，路径保持一致。
2. 配置环境变量

```
export QT_DIR=/opt/qt-everywhere-src-5.15.10/ext/
export LD_LIBRARY_PATH=${QT_DIR}/lib:${QT_DIR}/plugins/platforms
export QT_QPA_PLATFORM_PLUGIN_PATH=${QT_DIR}/plugins/platforms
export QML2_IMPORT_PATH=${QT_DIR}/qml
export QT_QPA_PLATFORM="xcb"
```

可以写到~/.bashrc文件里

2.7. 交叉编译QT工程

如有以下工程目录：

```
pg@pg-Default-string:~/qtest$ tree
.
├── ImageReceiver.cpp
├── ImageReceiver.h
├── main.cpp
└── main.pro
```

首先需要修改qt工程的pro文件，以下的文件为例：

```
TEMPLATE = app
TARGET = ImageReceiver #名称
QT += core gui network   widgets #依赖的qt库

QMAKE_CXX = aarch64-linux-gnu-g++ #指定g++路径
QMAKE_CC  = aarch64-linux-gnu-gcc #指定g++路径
QMAKE_LINK = aarch64-linux-gnu-g++ #指定g++路径

HEADERS += ImageReceiver.h #头文件
SOURCES += main.cpp ImageReceiver.cpp #源文件
```

然后在工程目录下执行：

```
pg@pg-Default-string:~/qtest$ /opt/qt-everywhere-src-5.15.10/host/bin/qmake .
```

会在目录下生成Makefile文件，如下所示为部分代码段：

```
#############################################################################
# Makefile for building: ImageReceiver
# Generated by qmake (3.1) (Qt 5.15.10)
# Project:  main.pro
# Template: app
# Command: /opt/qt-everywhere-src-5.15.10/host/bin/qmake -o Makefile main.pro
#############################################################################

MAKEFILE      = Makefile

EQ            = =

####### Compiler, tools and options

CC            = aarch64-linux-gnu-gcc
CXX           = aarch64-linux-gnu-g++
DEFINES       = -DQT_NO_DEBUG -DQT_WIDGETS_LIB -DQT_GUI_LIB -DQT_NETWORK_LIB -
DQT_CORE_LIB
CFLAGS        = -march=armv8-a --sysroot=/home/pg/orangepi/sysroot -O2 -Wall -
Wextra -D_REENTRANT -fPIC $(DEFINES)
CXXFLAGS      = -march=armv8-a --sysroot=/home/pg/orangepi/sysroot -O2 -Wall -
Wextra -D_REENTRANT -fPIC $(DEFINES)
```

```
INCPATH       = -I. -I/opt/qt-everywhere-src-5.15.10/ext/include -I/opt/qt-
everywhere-src-5.15.10/ext/include/QtWidgets -I/opt/qt-everywhere-src-
5.15.10/ext/include/QtGui -I/opt/qt-everywhere-src-5.15.10/ext/include/QtNetwork
-I/opt/qt-everywhere-src-5.15.10/ext/include/QtCore -I. -
I../orangepi/sysroot/usr/include -I../orangepi/sysroot/usr/include/aarch64-linux-
gnu -I/opt/qt-everywhere-src-5.15.10/host/mkspecs/devices/linux-orangepi-g++
QMAKE         = /opt/qt-everywhere-src-5.15.10/host/bin/qmake


。。。。。。
```

然后在工程目录下执行gmake进行编译

```
gmake
```

即可将编译出来的文件拷贝到arm开发板上去运行