目前网上的大多数 QT 移植教程还都停留在 qt4.8 版本，或者还有更老的 Qtopia ,但是目前 Qt 已经发展到最新的 5.7 版本了，我个人也已经使用了很长一段时间的 qt5.6 for windows ，本文就来介绍一下QT在嵌入式环境的搭建。
以JZ2440 为例，使用韦老大提供的 ubuntu9.10 虚拟机作为移植环境。当然，其它虚拟机也是可以的，我同样在 ubuntu14.04 移植成功，只不过需要额外装一些库，后面我会简单提及。
此外，我尝试使用 4.3.2 版本的交叉编译工具编译 Qt5.6 ,很多错误，编译成功之后使用 qt 编译应用程序时也存在问题，因此我是用的是友善提供的 4.4.3 版本的交叉编译工具。建议新手使用与我相同的编译环境。
因为我们更换了编译工具，因此，我们有接下来的4个工作。
    1、重新编译内核
    2、制作文件系统
    3、移植tslib

    4、移植qt

**教程涉及的资料：**
1、交叉编译器
  arm-[**Linux**](http://lib.csdn.net/base/linux)-gcc-4.4.3.tar.gz        .链接：http://pan.baidu.com/s/1nvJF8ud 密码：oi57
2、针对于 JZ2440 的内核补丁以及源码
  linux-2.6.22.6.tar.bz2           .链接：http://pan.baidu.com/s/1gf0oZn1 密码：jrp6
  linux-2.6.22.6\_jz2440.patch        链接：http://pan.baidu.com/s/1c1CWgTE 密码：uwuu
  4.3寸LCD\_mach-smdk2440.c          链接：http://pan.baidu.com/s/1boZsKwf 密码：1xml
3、busybox
  busybox-1.22.1.tar.bz2           .链接：http://pan.baidu.com/s/1cee6CI .密码：lv81
4、tslib                      .
  tslib-1.4.tar.gz
5、qt5.6 源码                      链接：http://pan.baidu.com/s/1jINj3IQ 密码：6kkc
  qt-everywhere-opensource-src-5.6.0.tar.gz 链接：http://pan.baidu.com/s/1kVnV9oN 密码：r7cm

**一、更换交叉编译器**

  首先，将下载好的文件全部放在 /work 目录下
1、解压交叉编译器
  sudo tar zxvf arm-linux-gcc-4.4.3.tar.gz -C /     #展开在根目录
  ls /opt/FriendlyARM/toolschain/4.4.3/bin        #检查是否生成了编译目录
2、设置环境变量
  sudo vi /etc/environment
  将现在原有的交叉编译器路径替换为我们新解压的交叉编译器，举例：

1. #PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/work/tools/gcc-3.4.5-glibc-2.3.6/bin"//注释
2. PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/opt/FriendlyARM/toolschain/4.4.3/bin" //增加

后期**[测试](http://lib.csdn.net/base/softwaretest%22%20%5Co%20%22%E8%BD%AF%E4%BB%B6%E6%B5%8B%E8%AF%95%E7%9F%A5%E8%AF%86%E5%BA%93%22%20%5Ct%20%22_blank)**时我发现这里是有问题的，由于我的安装目录都指定的是 usr 目录，需要 root 权限，然而在 environment 中设置的环境变量导致使用 root 权限 make install 过程中出现了个别的 command not found ，于是我将环境变量放到了 /etc/profile 中，这个文件是对所有用户生效的。至于environment, 百度了一下是系统环境变量按说应该没问题，但是的确出现了问题。索性还是大家还是直接搞到 profile 中吧。

3、重启虚拟机

4、检查是否更换成功

  arm-linux-gcc -v



如图，更换完毕，我们的编译器就变为 4.4.3 版本了。

**二、重新编译内核**

如果你不是jz2440也没关系，使用你曾经移植好的内核，重新编译即可，注意内核需要开启EABI的支持，如果你是jz2440，按照下面的步骤操作即可。

1、解压
  tar jxvf linux-2.6.22.6.tar.bz2
2、打补丁
  cd linux-2.6.22.6
  patch -p1 < ../linux-2.6.22.6\_jz2440.patch
  如果你是 4.3 寸的 JZ2440 ，替换 LCD 驱动，3.5寸跳过此步
  mv ../4.3寸LCD\_mach-smdk2440.c arch/arm/mach-s3c2440/mach-smdk2440.c
3、配置内核支持 EABI
  make menuconfig
  kernel features->
    <\*>use the arm eabi to .... //选中这一项
4、修改 makefile ，不改的话 4.4.3 编译器编译时会报错
  vi Makefile

  由于友善在制作这款交叉编译工具时指定了最低内核版本（2.6.32.2）的限制，所以我们如果不加修改直接编译启动内核时会出现 kernel too old 的错误信息，无法启动。这里使用一个投机取巧的办法，强制对2.6.22.6 内核的版本进行修改，更改内核版本号对内核编译是有影响的，真正移植的时候还是建议选择高版本内核。

  Makefile 最开始的地方就是版本号：
    VERSION = 2
    PATCHLEVEL = 6
    SUBLEVEL = 22
    EXTRAVERSION = .6
  中的 SUBLEVEL = 22 改为 SUBLEVEL = 32
  //找到一下几行 ，这里是因为 4.4.3 的编译器不支持 gdwarf2 这个选项
    ifdef CONFIG\_DEBUG\_INFO
    CFLAGS += -g
    AFLAGS += -gdwarf2
    endif
  //注释掉 AFLAGS += -gdwarf2
5、编译
  make uImage -j4
  编译完的uImage 在 /arch/arm/boot 目录下

**三、制作文件系统**

1、解压
  tar jxvf busybox-1.22.1.tar.bz2
2、配置
  make menuconfig
  Busybox Settings ->
  general configuration ->
    [\*] don't use /usr                   //选中它 否则会破坏虚拟机
  build options->
    cross comliler prefix = arm-linux-        //选择交叉编译工具
  installtion options->
    busybox installation prefix = /work/my\_rootfs //指定安装路径
3、编译&安装
  mkdir -p /work/my\_rootfs                 //安装路径
  make
  make install
  cd /work/my\_rootfs
  ls                 //查看是否安装成功
  编译安装完毕之后，我们的文件系统就生成在 /work/my\_roofs 目录下了
4、创建 /etc/inittab
  mkdir -p /work/my\_rootfs/etc
  vi /work/my\_rootfs/etc/inittab
  # /etc/inittab
  # 启动脚本/etc/init.d/rcS
  ::sysinit:/etc/init.d/rcS
  # 启动 shell
  ::askfirst:-/bin/sh
  # 重启关机前 卸载文件系统
  ::ctrlaltdel:/sbin/reboot
  ::shutdown:/bin/umount -a -r
5、创建 /etc/init.d/rcS
  mkdir -p /work/my\_rootfs/etc/init.d
  vi /work/my\_rootfs/etc/init.d/rcS
  # 这是一个脚本文件，用/bin/sh解析
  #!/bin/sh
  # 挂载文件系统
  mount -a
  # 使用内存文件系统
  mkdir /dev/pts
  mount -t devpts devpts /dev/pts
  echo /sbin/mdev > /proc/sys/kernel/hotplug
  mdev -s
  # 设置IP
  #/sbin/ifconfig eth0 192.168.1.17 #nfs不需要
  # 挂载 /etc/fstab 中的文件系统
  mount -a
  exec /etc/rc.local
6、创建rc.local
  vi /work/my\_rootfs/etc/rc.local
  #!/bin/sh
  . /etc/profile //注意.后边有个空格！
7、创建/etc/fstab
  vi /work/my\_rootfs/etc/fstab
  # device mount-point type options dump fsck order
  proc  /proc proc defaults 0 0
  sysfs  /sys sysfs defaults 0 0
  tmpfs /tmp tmpfs defaults 0 0
  tmpfs /dev tmpfs defaults 0 0
8、构建 /dev 目录
  mkdir -p /work/my\_rootfs/dev
  cd /work/my\_rootfs/dev
  sudo mknod console c 5 1
  sudo mknod null   c 1 3
9、创建其它目录
  mkdir proc mnt tmp sys root usr
10、拷贝Lib
  cd /opt/FriendlyARM/toolschain/4.4.3/arm-none-linux-gnueabi/lib
  cp \*.so\* /work/my\_rootfs/lib -d
  cd /opt/FriendlyARM/toolschain/4.4.3/lib
  cp \*.so\* /work/my\_rootfs/lib -d

**四、移植 tslib**

1、解压&配置&编译
  如果不是使用的韦老大的虚拟机可能还需要安装一下几个包autoconf automake libtool
  #sudo apt-get install autoconf automake libtool
  tar zxvf tslib-1.4.tar.gz
  sudo mkdir -p /usr/local/tslib
  cd tslib
  ./autogen.sh
  ./configure --host=arm-linux ac\_cv\_func\_malloc\_0\_nonnull=yes CC=arm-none-linux-gnueabi-gcc CXX=arm-none-linux-gnueabi-g++ -prefix=/usr/local/tslib
  make
  sudo make install

  如果编译过程中遇到 undefined reference to 'rpl\_malloc'，前面配置完成之后修改 config.h.in 文件，注释掉文件最后的 #undef malloc ，重新 make 即可。
  安装完成之后，tslib 就安装在虚拟机 /usr/local/tslib 目录下
2、更改 tslib 配置文件
  cd /usr/local/tslib/etc/
  sudo vi ts.conf
  去掉# module\_raw input 前面的 “#” 和空格
3、将制作好的 tslib 移动到我们制作的文件系统
  cd /usr/local
  sudo tar zcvf tslib.tar.gz tslib
  mkdir -p /work/my\_rootfs/usr/local
  cp tslib.tar.gz /work/my\_rootfs/usr/local
  tar zxvf tslib.tar.gz
  rm tslib.tar.gz
4、添加 tslib 环境变量
  vi /work/my\_rootfs/etc/profile

  #!/bin/sh
  export T\_ROOT=/usr/local/tslib
  export LD\_LIBRARY\_PATH=/usr/local/tslib/lib:$LD\_LIBRARY\_PATH
  export TSLIB\_CONSOLEDEVICE=none
  export TSLIB\_FBDEVICE=/dev/fb0
  export TSLIB\_TSDEVICE=/dev/input/event0
  export TSLIB\_PLUGINDIR=$T\_ROOT/lib/ts
  export TSLIB\_CONFFILE=$T\_ROOT/etc/ts.conf
  export POINTERCAL\_FILE=/etc/pointercal
  export TSLIB\_CALIBFILE=/etc/pointercal
  此时，tslib 就已经移植好了，你可以挂载 nfs 文件系统启动，cd /usr/local/tslib/bin

  ./ts\_test 来测试
  按照教程制作应该没有问题，我在使用高版本内核的时候遇到“selected device is not a touchscreen I understand”错误，百度了一下，是由于内核和编译器的一个宏定义不一致导致的，vi include/Linux/input.h  EV\_VERSION 为 0x0100000 ，从新编译内核特别是触摸驱动。

**五、移植 qt5.6**

1、解压
  tar zxvf qt-everywhere-opensource-src-5.6.0.tar
2、修改编译配置
  cd /work/qt-everywhere-opensource-src-5.6.0/qtbase/mkspecs/linux-arm-gnueabi-g++
  vi qmake.conf
针对于 2440 增加：
  QT\_QPA\_DEFAULT\_PLATFORM = linuxfb
  QMAKE\_CFLAGS += -msoft-float -D\_\_GCC\_FLOAT\_NOT\_NEEDED -march=armv4t -mtune=arm920t
  QMAKE\_CXXFLAGS += -msoft-float -D\_\_GCC\_FLOAT\_NOT\_NEEDED -march=armv4t -mtune=arm920t
  march 指的 cpu [**架构**](http://lib.csdn.net/base/architecture)，针对 2440 来说是 armv4t
  mtune 指的 cpu 名字，针对 2440 来说是 arm920t
  如果你是 A8 的板子 ，可以参考下边的配置
  QT\_QPA\_DEFAULT\_PLATFORM = linuxfb
  QMAKE\_CFLAGS  += -msoft-float -D\_\_GCC\_FLOAT\_NOT\_NEEDED -march=armv7-a -mtune=cortex-a8
  QMAKE\_CXXFLAGS += -msoft-float -D\_\_GCC\_FLOAT\_NOT\_NEEDED -march=armv7-a -mtune=cortex-a8
  如果你是 A9 的板子 ，可以参考下边的配置
  QT\_QPA\_DEFAULT\_PLATFORM = linuxfb
  QMAKE\_CFLAGS  += -msoft-float -D\_\_GCC\_FLOAT\_NOT\_NEEDED -march=armv7-a -mtune=cortex-a9
  QMAKE\_CXXFLAGS += -msoft-float -D\_\_GCC\_FLOAT\_NOT\_NEEDED -march=armv7-a -mtune=cortex-a9
将以下部分
  # modifications to g++.conf
  QMAKE\_CC = arm-linux-gnueabi-gcc
  QMAKE\_CXX = arm-linux-gnueabi-g++
  QMAKE\_LINK = arm-linux-gnueabi-g++
  QMAKE\_LINK\_SHLIB = arm-linux-gnueabi-g++

  # modifications to linux.conf
  QMAKE\_AR = arm-linux-gnueabi-ar cqs
  QMAKE\_OBJCOPY = arm-linux-gnueabi-objcopy
  QMAKE\_NM = arm-linux-gnueabi-nm -P
  QMAKE\_STRIP = arm-linux-gnueabi-strip
修改为：-lts 是指在链接时链接 tslib 库
  # modifications to g++.conf
  QMAKE\_CC = arm-none-linux-gnueabi-gcc -lts
  QMAKE\_CXX = arm-none-linux-gnueabi-g++ -lts
  QMAKE\_LINK = arm-none-linux-gnueabi-g++ -lts
  QMAKE\_LINK\_SHLIB = arm-none-linux-gnueabi-g++ -lts

  # modifications to linux.conf
  QMAKE\_AR = arm-none-linux-gnueabi-ar cqs
  QMAKE\_OBJCOPY = arm-none-linux-gnueabi-objcopy
  QMAKE\_NM = arm-none-linux-gnueabi-nm -P
  QMAKE\_STRIP = arm-none-linux-gnueabi-strip
3、配置编译
  sudo mkdir -p /usr/local/Qt5.6
  cd ../../../

  ./configure  -prefix /usr/local/Qt5.6 \
  -opensource \
  -release \
  -confirm-license \
  -xplatform linux-arm-gnueabi-g++ \
  -shared \
  -qt-zlib \
  -no-gif \
  -qt-libjpeg \
  -no-nis \
  -no-opengl \
  -no-cups \
  -no-glib \
  -no-dbus \
  -no-rpath \
  -no-sse2 -no-sse3 -no-ssse3 -no-sse4.1 -no-sse4.2 \
  -no-avx  \
  -no-openssl \
  -nomake tools \
  -qreal float \
  -qt-libpng \
  -tslib \
  -nomake examples \
  -I /usr/local/tslib/include \
  -L /usr/local/tslib/lib

  make -j4
  sudo make install

  如果使用的不是韦老大的虚拟机编译过程中可能报关于libxcb的错误，查看 qtbase/src/plugins/platforms/xcb 底下的 readme 安装相应的库就可以了。
4、将移植好的 qt 打包到开发板
  cd /usr/local
  sudo tar zcvf Qt5.6.tar.gz Qt5.6
  cp Qt5.6.tar.gz /work/my\_rootfs/usr/local/
  tar zxvf Qt5.6.tar.gz
  rm Qt5.6.tar.gz
  rm -r doc include bin mkspecs qml translations
5、设置qt相关的环境变量
  此部分可以参考qt官方问文档：http://doc.qt.io/qt-5/embedded-linux.html ，这我这仅仅是设置支持了触摸屏，你可以参考官方设置支持键盘，鼠标等等。

  在文件系统 /etc/profile 里添加
  export QTEDIR=/usr/local/Qt5.6
  export LD\_LIBRARY\_PATH=/usr/local/Qt5.6/lib:$LD\_LIBRARY\_PATH
  export QT\_QPA\_GENERIC\_PLUGINS=tslib
  export QT\_QPA\_FONTDIR=$QTEDIR/lib/fonts
  export QT\_QPA\_PLATFORM\_PLUGIN\_PATH=$QTEDIR/plugins
  export QT\_QPA\_PLATFORM=linuxfb:fb=/dev/fb0:size=480x272:mmSize=480x272:offset=0x0:tty=/dev/tty1

  export QT\_QPA\_FB\_TSLIB=1

  红色部分请根据自己的开发板进行修改，我手头的开发板是 3.5 寸 320\*240 的 LCD，JZ2440 V3 用上面的参数即可。

  此时，qt 已经移植完毕，你可以打包放入你的 nfs 目录启动进行测试了，至于制作 yaffs2 jffs2 等文件系统请参考：http://blog.csdn[**.NET**](http://lib.csdn.net/base/dotnet)/lizuobin2/article/details/52589215 ，qt库比较大，烧录的时候可能比较困难，可以先将QT去除，打包成文件系统大约之后20M不到，烧录到开发板之后，启动内核，通过nfs tftp等工具，再将打包好的Qt传到板子上展开即可。

**六、测试Qt应用程序**

  我个人习惯先用 windows 版本的 qt 写好程序，调试没问题之后直接拿到虚拟机上编译，windows 版本的 qt 安装非常简单，就跟安装个QQ一样毫不费力，这里就不再赘述了。

  将windows下编辑好的源码拷贝到虚拟机上，切换到源码目录

  /usr/local/Qt5.6/bin/qmake

  make

  即可生成可执行文件，拿到开发板测试即可。

  嫌费事的话，将 qmake 添加到环境变量里。

  在测试过程中我发现程序跑起来没问题，但是有以下两条错误信息：

  QIconvCodec::convertFromUnicode: using Latin-1 for conversion, iconv\_open failed
  QIconvCodec::convertToUnicode: using Latin-1 for conversion, iconv\_open failed

  大概是缺少 libiconv

  下载 链接：http://pan.baidu.com/s/1c22xb4O 密码：pbld

  mkdir -p /usr/local/libiconv

  ./configure --host=arm-none-linux-gnueabi --prefix=/usr/local/libiconv CC=arm-none-linux-gnueabi-gcc LDFLAGS="-L/opt/FriendlyARM/toolschain/4.4.3/arm-none-linux-gnueabi/sys-root/lib" --enable-static

  make

  sudo make install

  把安装目录/lib 下的 preloadable\_libiconv.so 文件系统的 /lib 下, 在 /etc/profile 中添加
  export LD\_PRELOAD=/lib/preloadable\_libiconv.so

**7、使Qt支持中文**

  在测试过程中很悲剧的发现移植好的qt不支持中文，强大度娘给了我答案。

  下载字库 链接：http://pan.baidu.com/s/1bp9QFQv 密码：2u81

  将DroidSansFallback.ttf 放到文件系统 /usr/local/Qt5.6/lib/fonts 目录下，虽然Qt自带了很多字库了，但是都没有中文的。

 然后在使用qt creator 开发qt应用程序时，在项目中添加

  QT       += gui

  包含头文件 #include <QFontDatabase>

  代码中设置使用的字体。

 int id = QFontDatabase::addApplicationFont("/usr/local/Qt5.6/lib/fonts/DroidSansFallback.ttf");

 QString msyh = QFontDatabase::applicationFontFamilies (id).at(0);

 QFont font(msyh,10);

 font.setPointSize(20);

 this->setFont(font);



**8、制作好的文件系统&内核**

  8.1 简单的qt例子，代码中Ui尺寸写死为 320\*240 ，如果不适合你的板子请修改。

   链接：http://pan.baidu.com/s/1qXV4C1Y 密码：908q

  8.2 jz2440 内核（包含3.5和4.3）

   链接：http://pan.baidu.com/s/1qYMrxmc 密码：qsk0

  8.3 文件系统

   链接：http://pan.baidu.com/s/1skLtj3J 密码：lbo2

   改文件系统没有打包成镜像，请使用nfs启动，解压之后在 dev 目录创建 console null 设备节点。

   此外，/etc/profile 环境变量中的参数请修改成符合自己开发板的参数，默认为3.5寸。

   解压缩之后80多M，制作成 yaffs 恐怕会过 100 M，qt的库文件没有进行剪裁，需要制作镜像的同学可以进行剪裁（查看你的app依赖于哪些库，其余的都可以去掉）。

如果你想让你的开发板支持 ssh 请参考：

  http://blog.csdn[**.Net**](http://lib.csdn.net/base/dotnet)/lizuobin2/article/details/52664339

如果你想让你的开发板支持串口（rz）传输文件请参考：

  http://blog.csdn.net/lizuobin2/article/details/52601617

如果你想让你的开发板想虚拟机一样在命令行显示用户名请参考：

  http://blog.csdn.net/lizuobin2/article/details/52664657