Android 驱动开发

树莓派 3

版本	描述	日期
V1.0	创建	2014.1.25

微信公众号

更多技术支持与科技资讯分享，请关注我们的微信公众号。

目录

版权声明

敬告：

在售产品的手册会经常更新，请在 www.SuperPi.org 网站下载最新手册，不再另行通知。

如果您对文档有所疑问，您可以在办公时间（星期一至星期五上午 9:00~12:00；下午 1:30~6:00；星期六上午 9:00~12:00）到论坛发贴或 E-mail 联系我们。

网址：www.SuperPi.org

联系电话：0755-23012315

E-mail：peter@arm-cn.com

销售及网络：

深圳风火轮科技有限公司

电话：0755-23012315 - 18665399520

传真：0755-23081825

地址：深圳市宝安区宝源路名优工业产品展示采购中心 C 座 710、711 号

网址：www.SmartFire.cn

深圳本地客户可以上门提货

树莓派淘宝店：http://supay.taobao.com/ --专营树莓派

风火轮淘宝店：http://SmartFire.taobao.com/ --专营各类开发板

技术支持：

树莓派论坛：www.SuperPi.org

QQ 群: 237172995

QQ 群: 249126495

QQ 群: 178954903

QQ 群: 209767435

(由于资源有限，请只选其中一个 QQ 群加入，多加无用)

风火轮推荐的技术支持方式是论坛发贴，我们的工程师会第一时间去回贴。

摘要

本文主要介绍树莓派 3 的 android 系统驱动开发指导

更多技术文档，关注树莓派官网 www.SuperPi.org

作为 **Web server 服务器**
作为 **Android TV 机顶盒**
作为 **NAS 服务器**
作为**控制主板，机器的核心**
作为**课堂学生的实验板**
作为**产品预研的评估板**
…….

关键字: 嵌入式,树莓派, android,裁剪,编译,

第1章 android驱动的开发流程

1： 写 LINUX 驱动

2： 写 LINUX 应用测试程序

3： 写 JNI 接口，用来包装第二步写的应用

（要用 NDK 来编译）

生成一个.SO 文件，相当于 CE 下的 DLL

4： 写 JAVA 程序,专门写一个类包含.SO 文件，然后在 JAVA 里调用.SO 里的函数。

例子，可以看 NDK 里面的 Sample 文件夹，里面有一些例子

第2章 开发环境搭建

基于 windows 系统作为开发环境！

编译 Android 的 LINUX 交叉编译工具（在 LINUX 系统下）

编译 LINUX 驱动的交叉编译工具(4.3.1) （在 LINUX 系统下）

编译 JNI 的工具包：NDK（在 LINUX 系统下）

编译 JAVA 程序：esclips+ADT+SDK（在 windows 系统下）

2.1 NDK安装

2.1.1 下载NDK包

（**本节在 LINUX 系统下操作**）

下载地址：

http://dl.google.com/android/ndk/android-ndk-r4b-linux-x86.zip

2.1.2 解压

到/home/workspace/目录

2.1.3 编辑环境变量

sudo gedit /etc/profile

在末行加入

#set NDK env

NDKROOT=/home/workspace/android-ndk-r4b

export PATH=$NDKROOT:$PATH

更新修改

source /etc/profile

此时，系统就能识别 ndk-build 命令了

2.1.4 编译例子

进入 sampleshello-jni 目录，编译

```
cd samples/hello-jni
aulyp@ubuntu:/home/workspace/android-ndk-r4b/samples/hello-jni$ ndk-build
```

就能看到编译信息了，如果编译成功，会在该目录多生成 2 个子目录 libs,obj 目录

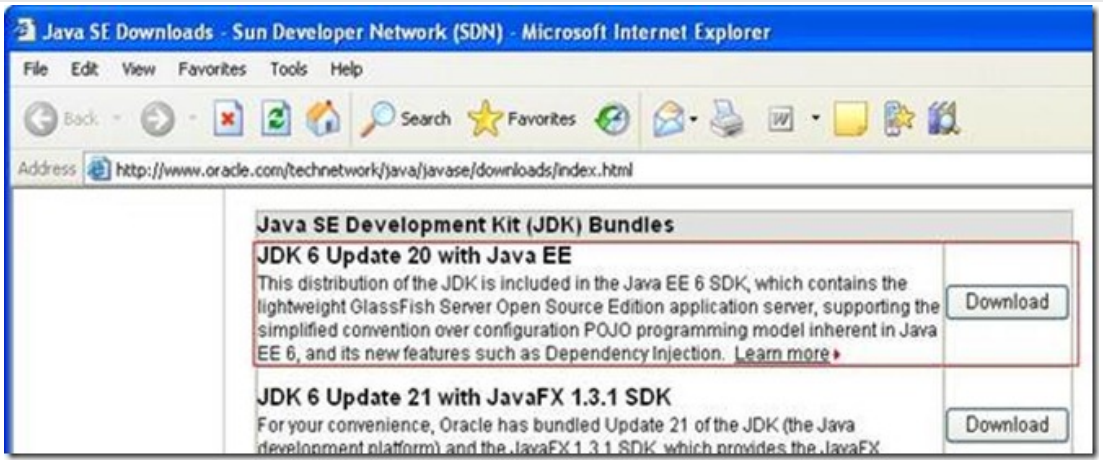
2.2 JDK安装

（**本节在 Windows 系统下操作**）

到 Sun 官方网站下载 JDK6,选择 JDK 6 Update 20

下载页面地址： http://www.oracle.com/technetwork/java/javase/downloads/index.html

INCLUDEPICTURE "http://www.buddyinfo.com.cn/images/Android/android2.jpg" /* MERGEFORMATINET
INCLUDEPICTURE "http://www.buddyinfo.com.cn/images/Android/android2.jpg" /* MERGEFORMATINET
INCLUDEPICTURE "http://www.buddyinfo.com.cn/images/Android/android2.jpg" /* MERGEFORMATINET
INCLUDEPICTURE "http://www.buddyinfo.com.cn/images/Android/android2.jpg" /* MERGEFORMATINET
INCLUDEPICTURE "http://www.buddyinfo.com.cn/images/Android/android2.jpg" /* MERGEFORMATINET



下载完后，双击进行安装

安装成功，在 cmd 下输入 java –version，会有 JAVA 的版本信息出来

```
C:/Users/Aulyp>java -version
java version "1.6.0_21"
Java(TM) SE Runtime Environment (build 1.6.0_21-b07)
Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
```

2.3 Esclips安装

（本节在 LINUX 或 windows 系统下操作，看你喜欢在哪个系统都行）

2.3.1 Linux:

官方下载：http://www.eclipse.org/downloads/

到 Eclipse 官方网站下载 Ecplise For Java EE 的最新版本

我选择下载的是 eclipse-java-helios-SR1-linux-gtk.tar

解压到指定目录：/opt

```
aulyp@ubuntu:/opt$ sudo tar zxvf eclipse-java-helios-SR1-linux-gtk.tar.gz
```

得到 eclipse 文件夹

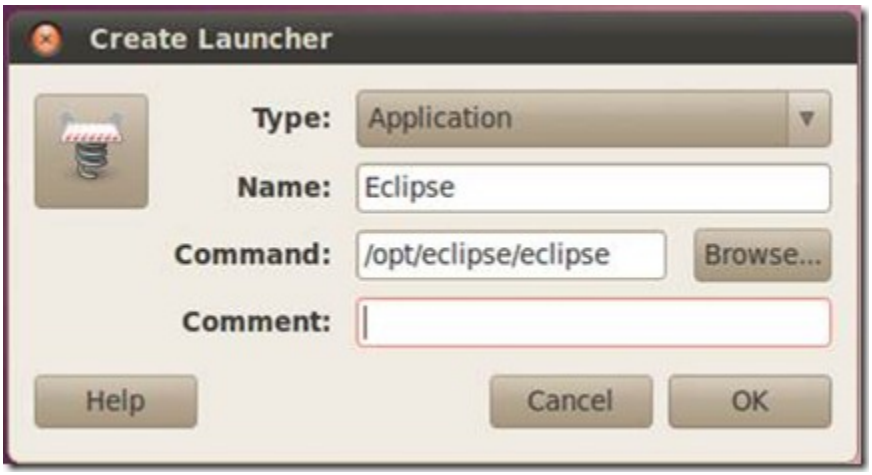
在桌面上创建启动图标；

在桌面（右键单击桌面->创建启动器）；然后选择

名称：Eclipse

命令：eclipse （点 Browse 进去选择）

图标： /opt/eclipse/icon.xpm（Ubuntu 10.04 上面有个图标，点击之后，选择路径）



在桌面得到一个图标，这样可以双击该图标，打开 Eclipse。

2.3.2 Windows

下载后，直接解压，就能运行了。

2.4 ADT安装和配置

（**本节在 Windows 系统下操作**）

安装和配置 ADT

ADT=Android Develop Tools .

(1)启动安装完(解压)后的 Ecplise 目录下的 Ecplise.exe,启动 Eclips

(2)点击菜单"Help"->"Install New Software"

(3)选择以下地址: http://dl-ssl.google.com/android/eclipse/ 然后点击"OK"

(4)选择 Developer Tools"进行 ADT(Android Development Tools)的安装工作.

(5)Ecplise 此时会从加入的地址中自动下载跟安装 ADT,稍等一会,安装完毕后,提示需要重启 Ecplise,点击"OK"重启 Ecplise

如果出现错误，请把 Https 改成 Http，也就是换一个协议地址来安装。

再接下来的对话框中选中所有的 Name 下的所有选项。然后根据提示来完成后续的安装过程。

最后一步，关联 Android SDK 和 Eclipse

打开菜单 Windows,Referces,Android。点击“Browse”选择 Android SDK 的安装路径。点击 OK。

检测是否安装正确：

菜单 File,New,Project，出现 AndroidProject 选项。说明安装配置成功。

2.5 SDK安装

（本节在 Windows 系统下操作）

download android sdk 2.1

官方下载地址：http://developer.android.com/sdk/index.html

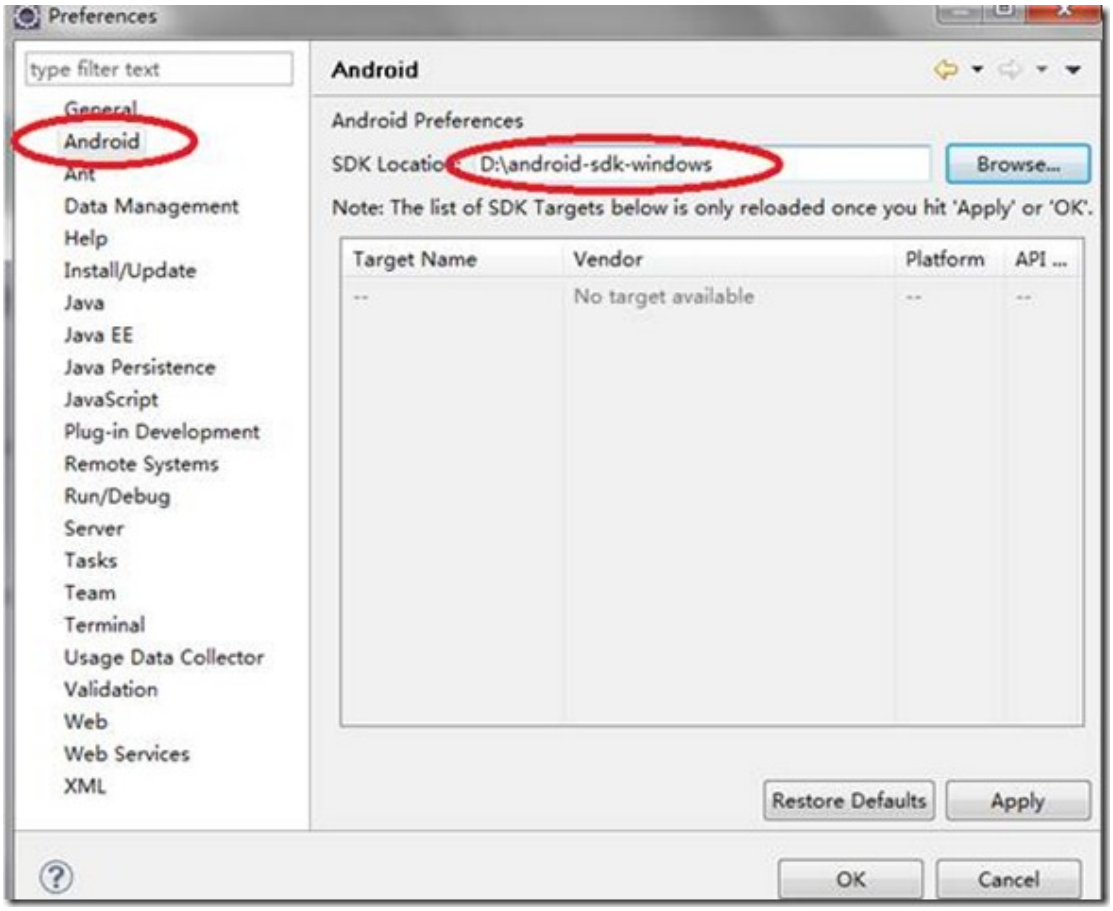
但是，如果不用代理，是上不了官网的，可以直接用下面的地址下载

r07 版本: http://dl.google.com/android/android-sdk_r07-windows.zip

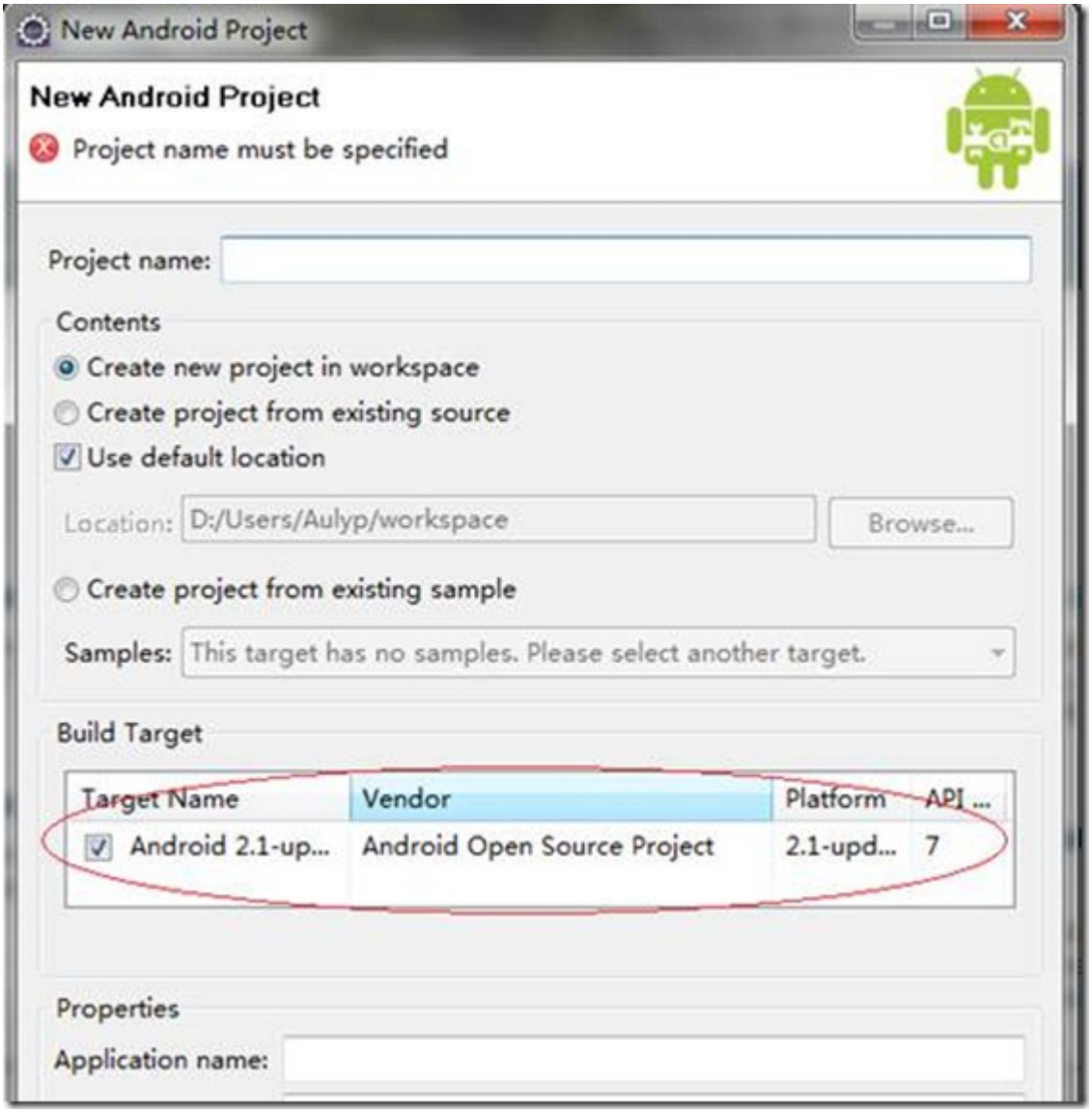
r04 版本:http://dl.google.com/android/android-sdk_r04-windows.zip

解压在某一个目录下。

重启完 Ecplise 后,点击菜单"Window"->"Preference",打开了配置窗口,在左边选中 Android,在右边输入 Android SDK 的路径,点击"OK",至此环境配置完毕.



关联成功，在新建工程时，才能找得到 Android 的 SDK 的，如下图


New Android Project
New Android Project
Project name must be specified
Project name:
Contents
Create new project in workspace
Create project from existing source
Use default location
Location: D:/Users/Aulyp/workspace
Browse...
Create project from existing sample
Samples: This target has no samples. Please select another target.
Build Target
Target Name
Vendor
Platform
API ...
Android 2.1-up...
Android Open Source Project
2.1-upd...
7
Properties
Application name:

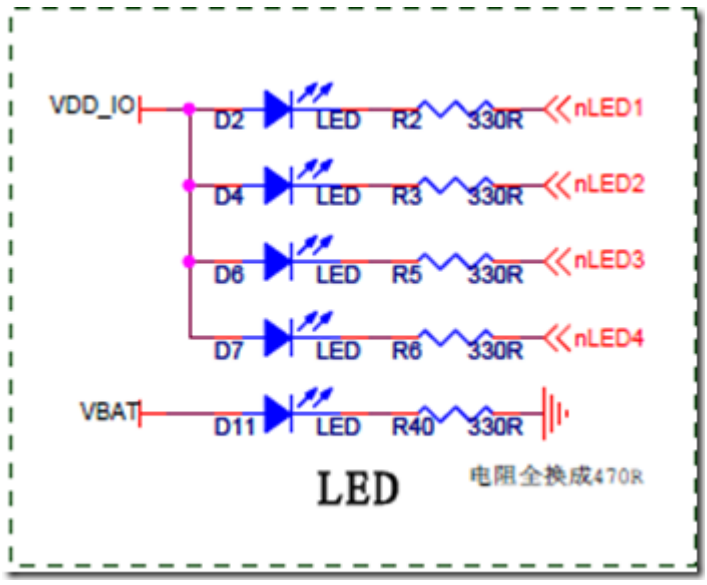
第3章 android驱动例子（LED灯控制）

本例子，是本人前几年基于 **tiny6410 开发板+android2.1 系统** 进行讲解的，其实在树莓派 3 上是几乎同样的，有极大的参考价值，依样画葫芦就可以开发出基于树莓派 3 的 android 驱动了。后续会有更多直接在树莓派上做和例子出来，请关注树莓派论坛 www.SuperPi.org

本例讲述在 android2.1 上完全自己开发一个驱动去控制硬件口并写应用测试该驱动，通过这样一个例子，解析 android 下的驱动开发流程的应用调用流程，可以说是很好的入门引导

要达到的效果：通过 android 的应用，调用驱动程序，在 Tiny6410 开发板上控制 4 个 LED 的亮灭。

3.1 硬件原理



如上图，通过 4 个 IO 口控制这 LED，低电平 LED 亮，

这 4 个 IO 口分别是 GPM1, GPM2, GPM3, GPM4,

3.2 编写驱动程序

1、在 kernel 文件夹下的 driver 目录，新键驱动文件夹

cd kernel_android_2.6.28.6/drivers

进到开发板的 kernel 目录，建驱动文件夹

```
#mkdir ledtest
```

2、在/driver/ledtest 目录下，新建 leddriver.c ，leddriver.h , Kconfig, Makefile 等 4 个文件

leddriver.c

```
#include <linux/kernel.h>
```

```
#include <linux/sched.h>
#include <linux/timer.h>
#include <linux/init.h>          /* For __init/__exit/... */
#include <linux/module.h>
#include <mach/hardware.h>
#include <asm/io.h>
#include <asm/uaccess.h>
#include <plat/gpio-cfg.h>
#include <plat/regs-clock.h>
#include <plat/regs-lcd.h>
#include <plat/regs-gpio.h>
#include <mach/map.h>
#include <linux/gpio.h>
#include <plat/gpio-bank-m.h>
#include <linux/cdev.h>
#include <linux/fs.h> //for register_chrdev()
#include <linux/device.h>
#include <mach/map.h>
#include "leddriver.h"
#include <linux/miscdevice.h>       /* For MODULE_ALIAS_MISCDEV
                        (WATCHDOG_MINOR) */
#include <linux/watchdog.h>       /* For the watchdog specific items */
#include <linux/fs.h>          /* For file operations */
#define Viberator_MAJOR       97  //主设备号
#define SCULL_NR_DEVS          4
#define SCULL_QUANTUM        4000
#define SCULL_QSET           1000
//---do as the GIO driver
#define DEVCOUNT               4
#define GIO_MINOR           2    /* GIO minor no. */
static dev_t dev;       //申  请  到的主  设备  号
static struct cdev *cdev_p;
static int openCnt;
//--全局变量------------
int VIB_major  = 97;//we asigment it for test
int VIB_minor = 0;
int VIB_nr_devs = SCULL_NR_DEVS;
int VIB_quantum = SCULL_QUANTUM;
int VIB_qset = SCULL_QSET;

static struct class *vib_dev_class;
#define GPNCON  S3C64XX_GPNCON
```

```
#define GPNDAT  S3C64XX_GPNDAT
#define GPNPUD  S3C64XX_GPNPUD
#define GPMCON S3C64XX_GPMCON
#define GPMDAT  S3C64XX_GPMDAT
#define GPMPUD S3C64XX_GPMPUD

#define VIB_ON 0x11
#define  VIB_OFF 0x22
static const struct file_operations GPIO_Viberator_ctl_ops={
    .owner       = THIS_MODULE,
    .open    = GPIO_VIB_open,
    .read    =GPIO_VIB_read,
    .write    =GPIO_VIB_write,
    .ioctl     = GPIO_VIB_ioctl,
    .release     =GPIO_VIB_release,
};
ssize_t GPIO_VIB_read(struct file * file,char * buf,size_t count,loff_t
* f_ops)
{
      char sdas[5]={1,3,8,1,6};
      ssize_t          status = 5;
    unsigned long    missing;
    missing = copy_to_user(buf, sdas, status);
    if (missing == status)
        status = -EFAULT;
    else
        status = status - missing;
       printk(" GPIO_VIB_read = %d ,count = %d/r/n",status,count);
    gpio_direction_output(S3C64XX_GPM(3), 0);//
return status ;
}
ssize_t GPIO_VIB_write (struct file * file,const char * buf, size_t
count,loff_t * f_ops)
{
    unsigned long missing;
     char inbuffer[512]={0};
    int i;
     missing = copy_from_user(inbuffer, buf, count);
    printk(" GPIO_VIB_write count = %d missing =%d /r/n",count,missing);
    gpio_direction_output(S3C64XX_GPM(3), 1);//
    for(i=0;i<count;i++)
        {
```

```
            printk("Write:inbuffer[%d]=%d /r/n",i,inbuffer[i]);
        }
return count;
}

//ssize_t GPIO_VIB_ioctl(struct inode *  inode,struct file *
file,unsigned int cmd, long data)
static int GPIO_VIB_ioctl(struct inode *inode, struct file *file,
unsigned int cmd, unsigned long arg)
{
    printk(KERN_ERR"VIB:GPIO_VIB_ioctl --CMD=%x /n",cmd);
    switch(cmd)
        {
          case VIB_ON:
            gpio_direction_output(S3C64XX_GPM(1), 0);// 注意，树莓派 3 的 IO
控制可能与这里有点区别，请参考树莓派 3 的 android 源码里的其他驱动去控制 IO。
            gpio_direction_output(S3C64XX_GPM(2), 0);//
            gpio_direction_output(S3C64XX_GPM(3), 0);//
            gpio_direction_output(S3C64XX_GPM(4), 1);//
            printk(KERN_ERR"VIB:GPIO_VIB_ioctl --VIB_ON/n");
            break;
         case VIB_OFF:
            {
            printk(KERN_ERR"VIB:GPIO_VIB_ioctl --VIB_OFF/n");
            gpio_direction_output(S3C64XX_GPM(1), 1);//
            gpio_direction_output(S3C64XX_GPM(2), 1);//
            gpio_direction_output(S3C64XX_GPM(3), 1);//
            gpio_direction_output(S3C64XX_GPM(4), 0);//
            break;
         }
         default:break;

    }
        //gpio_free(S3C64XX_GPN(7));
}
ssize_t GPIO_VIB_open(struct inode * inode,struct file * file)
{
//模块数
  //MOD_INC_USE_COUNT;
  printk("GPIO_VIB_open() /n");
  return 0;
}
```

```
ssize_t GPIO_VIB_release(struct inode * inode, struct file * file)
{
   // MOD_DEC_USE_COUNT;//模块数减 1
     printk("GPIO_VIB_release() /n");
    return 0;
}
static int GPIO_VIB_CTL_init(void)
{
    int ret  = -ENODEV;
   int error ;
    printk("---------------------------------------------- /r/n");
    //初始化端口
        s3c_gpio_cfgpin(S3C64XX_GPM(1), S3C_GPIO_SFN(1));//GPM1 output
        s3c_gpio_cfgpin(S3C64XX_GPM(2), S3C_GPIO_SFN(1));//GPM2 output
        s3c_gpio_cfgpin(S3C64XX_GPM(3), S3C_GPIO_SFN(1));//GPM3 output
        s3c_gpio_cfgpin(S3C64XX_GPM(4), S3C_GPIO_SFN(1));//GPM4 output
#if 1 /*静态方式注册驱动*/
ret = register_chrdev(Viberator_MAJOR, "viberator",
&GPIO_Viberator_ctl_ops);
if (ret < 0) {
    printk(KERN_ERR "VIB: unable to get major %d/n", ret);
    return ret;
}
//创建 class
vib_dev_class = class_create(THIS_MODULE, "viberator");
if (IS_ERR(vib_dev_class)) {
    unregister_chrdev(Viberator_MAJOR, "capi20");
    return PTR_ERR(vib_dev_class);
}
//创建节点，
device_create(vib_dev_class, NULL, MKDEV(Viberator_MAJOR, 0), NULL,
"vib");
// create a point under /dev/class/vib
//通过上面这两步，驱动加载后，就会在/dev/class/下面生成 vib 节点，应用程序可以像
操作文件那样操作这个节点，通过 open ,write,read 等函数操作，详情可以看后面的应用
示例程序。
  return 0;
#endif
#if 0/* 注册动态 */
    if ((error = alloc_chrdev_region(&dev, 0, DEVCOUNT, "vibrate")) < 0)
        {
        printk(KERN_ERR
```

```
            "VIB: Couldn't alloc_chrdev_region, error=%d/n",
            error);
        return 1;
    }
        printk("dev = %d  /n",dev);
    cdev_p = cdev_alloc();
    cdev_p->ops = &GPIO_Viberator_ctl_ops;
    error = cdev_add(cdev_p, dev, DEVCOUNT);
    if (error) {
        printk(KERN_ERR
            "VIB: Couldn't cdev_add, error=%d/n", error);
        return 1;
    }

    vib_dev_class = class_create(THIS_MODULE, "vib-dev");
    if (IS_ERR(vib_dev_class)) {
        res = PTR_ERR(vib_dev_class);
        goto out_unreg_class;
    }
    return 0;
#endif
out_unreg_class:
    class_destroy(vib_dev_class);
return 1;
}

static int __init S3C6410_VIB_init(void)
{
    int ret = -ENODEV;
    //调用函数
        printk(KERN_ERR "Auly: S3C6410_VIB_init---/n");
    ret = GPIO_VIB_CTL_init();
    if(ret)
        {
        printk(KERN_ERR "Auly: S3C6410_VIB_init--Fail !!!/n");
        return ret;
    }
    return 0;
}
static void __exit cleanup_GPIO_VIB(void)
{
    //注销设备
```

```
 //  devfs_unregister_chrdev(Viberator_MAJOR,"gpio_vib_ctl");
#if 0
    cdev_del(cdev_p);
   unregister_chrdev_region(dev, DEVCOUNT);
   class_destroy(vib_dev_class);
#endif
device_destroy(vib_dev_class, MKDEV(Viberator_MAJOR, 0));
class_destroy(vib_dev_class);
unregister_chrdev(Viberator_MAJOR, "viberator");
}
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Peter first driver");
MODULE_ALIAS_CHARDEV(Viberator_MAJOR, 0);

module_init(S3C6410_VIB_init);
module_exit(cleanup_GPIO_VIB);
```

leddriver.h 文件

leddriver.h 文件

```
ssize_t GPIO_VIB_read(struct file * file,char * buf,size_t count,loff_t * f_ops);
ssize_t GPIO_VIB_write (struct file * file,constchar * buf, size_t count,loff_t * f_ops);
static int GPIO_VIB_ioctl(struct inode *inode,struct file *file, unsigned int cmd, unsigned long arg);
ssize_t GPIO_VIB_open(struct inode * inode,struct file * file);
ssize_t GPIO_VIB_release(struct inode * inode,struct file * file);
static int GPIO_VIB_CTL_init(void);
```

Kconfig 文件

Kconfig 文件

```
config LEDTEST
 tristate "LED test for ARMeasy"
 default n
 help
 this is a LED driver for ARMEASY with S3C6410
```

Makefile 文件

Makefile 文件

```
obj-$(CONFIG_LEDTEST)+=leddriver.o
```

3、在内核配置里菜单里加入本驱动的配置项

达到的效果是:可以通过内核配置来选择是否把本驱动编译进内核里，也就是出现在 make menuconfig 后出来的界面里，

可以参考《android 添加新驱动方法》

1）在 arch/arm/Kconfig 文件里 menu "Device Drivers"与 endmenu 之间添加

```
source "drivers/ledtest/Kconfig"
```

2）在 drivers/Kconfig menu "Device Drivers" 和 endmenu 之间添加

```
source "drivers/ledtest/Kconfig"
```

3)修改/drivers/Makefile 文件

```
Obj-$(CONFIG_LEDTEST) +=ledtest/
```

3.3 编译驱动

在 kernel 目录下，终端输入

```
#make menuconfig
```

“Device Drivers” 下面会看到 “LED test for ARMeasy”，选择它，

保存并退出

```
#make
```

这样，就会在/drivers/ledtest 目录下得到 leddrivr.ko 文件，它就是驱动目标文件，已级编译进了 zImage 里了，只要用这个 kernel 烧录到开发板，开机就会自动加载本驱动，

PS，如果发现如上编译，没有在 ledtest 目录下生成 leddriver.ko 文件，也就是本根没有去编译本驱动，那就把 ledtest 目录入到/drivers/misc 目录下，相应的修改 misc 下面，然后，在 makefile 里，不用选直接写成 obj-y += leddriver.o，这样强制的包含进去编译。

编译后生成了 leddriver.ko 文件，这就是驱动程序文件。

3.4 写应用程序（用于测试驱动）

用 C 语言写一个应用，测试上面写的驱动是否工作正常，这是一般的驱动开发流程

主要有两个文件：Android.mk 和 ledtest.c

1、在$(YOUR_ANDROID)/external/LEDTEST/目录编写 ledtest.c 文件，

ledtest.c

ledtest.c

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>// contact the open(),close(),read(),write() and so on!
 #define DEVICE_NAME"/dev/vib"//device point
#define LED_ON      0x11
#define LED_OFF      0x22
int main(int argc,char **argv)
{
   int fd;
   int ret;
```

```
  char *i;
  printf("\n start gpio_led_driver test \r\n");
  fd = open(DEVICE_NAME,O_RDWR);//Open device ,get the handle
  printf("fd = %d \n",fd);
  if(fd == -1) //open fail
  {
    printf("open device %s error \n",DEVICE_NAME);
  }
  else
  {
     while(1)
     {
       ioctl(fd,LED_OFF); //call the output function to off LEDs
       sleep(1);//wait 1 second
       ioctl(fd,LED_ON);
       sleep(1);
     }
     ret = close(fd); //close device
     printf("ret = %d \n",ret);
     printf("close gpio_led test \n");
  }
 return 0;
}
```

2、在$(YOUR_ANDROID)/external/ LEDTEST /目录编写 Android.mk 文件。这是 Android Makefile 的标准命名，不要更改。Android.mk 文件的格式和内容可以参考其他已有的 Android.mk 文件的写法，针对 helloworld 程序的 Android.mk 文件内容如下：

Android.mk

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_SRC_FILES:= \par     hello.c
LOCAL_MODULE := ledtestapp
include $(BUILD_EXECUTABLE)
```

注意上面 LOCAL_SRC_FILES 用来指定源文件；

LOCAL_MODULE 指定要编译的模块的名字，下一步骤编译时就要用到；

include $(BUILD_EXECUTABLE)表示要编译成一个可执行文件，如果想编译成动态库则可用 BUILD_SHARED_LIBRARY，这些可以在 $(YOUR_ANDROID)/build/core/config.mk 查到。

3、 回到 Android 源代码顶层目录进行编译：

```
# cd $(YOUR_ANDROID) && make ledtestapp
```

例如我这里的实际操作是

```
#cd android2.1
#make ledtestapp
```

注意 make ledtestapp 中的目标名 ledtestapp 就是上面 Android.mk 文件中由 LOCAL_MODULE 指定的模块名。

编译结果如下

```
make ledtestapp
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=2.1-update1
TARGET_PRODUCT=generic
TARGET_BUILD_VARIANT=eng
TARGET_SIMULATOR=
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm
HOST_ARCH=x86
HOST_OS=linux
HOST_BUILD_TYPE=release
BUILD_ID=ECLAIR
============================================
/bin/bash: line 0: cd: sdk/layoutopt/app/src/resources: No such file or directory
target thumb C: ledtestapp <= external/LEDTEST/ledtest.c
Install: out/host/linux-x86/bin/apriori
Install: out/host/linux-x86/bin/soslim
target Executable: ledtestapp
(out/target/product/generic/obj/EXECUTABLES/ledtestapp_intermediates/LINKED/ledtestapp)
target Non-prelinked: ledtestapp (out/target/product/generic/symbols/system/bin/ledtestapp)
target Strip: ledtestapp (out/target/product/generic/obj/EXECUTABLES/ledtestapp_intermediates/ledtestapp)
Install: out/target/product/generic/system/lib/libdl.so
Install: out/target/product/generic/system/lib/libc.so
Install: out/target/product/generic/system/lib/libstdc++.so
Install: out/target/product/generic/system/lib/libm.so
Install: out/target/product/generic/system/bin/ledtestapp
```

这一步，会在 out/target/product/generic/system/bin/目录 可执行文件是 ledtestapp
这个就是测试程序，在 linux 下运行的，我们可以在串口命令行中运行它来测试我们的驱动是否正常工作。

3.5 驱动测试

1、手工加载驱动

adb 或是用 TF 卡拷贝编译好的驱动 leddriver.ko 到系统根目录下

串口终端，或是 adb 终端输入

```
insmod / leddriver.ko
```

手动加载驱动程序

2、这样在开发板上的 android 系统中，在目录/dev/下生成节点：/dev/vib

3、开放节点权限

```
chmod 777 /dev/vib
```

有了这一步，后面的应用程序才能打开这个节点，

4、如上面的编译结果所示，编译后的可执行文件存放在

out/target/product/generic/system/bin/目录 可执行文件是 ledtestapp

5、拷贝 ledtestapp 可执行文件到 T 卡，插入开发板

6、开电，在启动进入 android 的过程中，串口终端 按回车，进入命令行模式

7、拷贝可执行文件到根目录

```
#cp /sdcard/ledtestapp /
```

8、执行程序

```
#./ledtestapp
```

通过上面的步骤，可以看到开发板上相应的 LED 在闭，每秒钟闪一次！

第4章 Android通过JNI调用驱动程序(完全解析实例)

本例子，是针对 android2.1 写的，现在树莓派 3 的 android 系统是 4.2 的，但是方法也是一样的，只是在选 SDK 版本的时候，要选到 4.2，就这点区别，本章主要讲解的是 android 开发的流程和思路，这些在不同版本里是相通的。

4.1 要达到的效果：

android 系统中，用 JAVA 写界面程序，调用 jni 中间库提供的接口，去操作某个驱动节点，实现 read,writer ioctl 等操作！这对底层驱动开发人员是很重要的一个调试通道，也是 android 系统下提供一些特殊功能接口的方法！

本文前提：

我们假设已经写了一个驱动程序，它是控制 LED 的亮灭的，并且创建了一个节点：/dev/vib，也就是通过 open 这个 vib 节点，可以 read/write/ioctl 操作驱动程序实现 LED 灯的亮灭控制，具体可以看上一章的《android 驱动例子（LED 灯控制）》

开发环境

1、ubuntu 下的 NDK 编译环境，2、Esclips 开发环境

4.2 编写JNI模块

当安装好 NDK 编译环境后，会在它的目录下找到 sample 目录，它里面有一些例子，可以参考这些例子来写我们自己的模块。

1、 source 文件夹下，新建“LEDSJNI”文件夹。

2、 Source/LEDSJNI/jni/目录下，新建“vib-jni.c”

vib-jni.c 文件

```
#include <string.h>
#include <jni.h>
#include <fcntl.h> /*包括文件操作，如 open() read() close() write()等*/
//----for output the debug log message
#include <android/log.h>
#define  LOG_TAG    "vib-jni"
#define  LOGI(...)
```

```
__android_log_print(ANDROID_LOG_INFO,LOG_TAG,__VA_ARGS__)
#define  LOGE(...)
__android_log_print(ANDROID_LOG_ERROR,LOG_TAG,__VA_ARGS__)
#define  DEVICE_NAME  "/dev/vib"  //device point
#define  VIB_ON 0x11
#define  VIB_OFF 0x22
int fd;
jstring
Java_com_auly_control_vibClass_stringFromJNI( JNIEnv* env,
                                              jobject thiz )
{
    return (*env)->NewStringUTF(env, "Hello from JNI--Peter for vib!");//
打印字符串
}

jint
Java_com_auly_control_vibClass_Init( JNIEnv*  env )
{
   LOGE("vibClass_Init() /n");
   fd = open(DEVICE_NAME,O_RDWR);//打开设备
   LOGE("vibClass_Init()-> fd = %d  /n",fd);
   if(fd == -1)
      {
         LOGE("open device %s error /n ",DEVICE_NAME);//打印调试信息
         return 0;
   }
   else
      {
        return 1;
      }
}

jint
Java_com_auly_control_vibClass_IOCTLVIB( JNIEnv*  env, jobject  thiz,
jint controlcode )
{
    int CTLCODE = controlcode;
    LOGE("IOCTLVIB() = %x --vibClass_IOCTLVIB /n",CTLCODE);
      switch(CTLCODE)
         {
       case VIB_ON:
             {
```

```
                ioctl(fd,VIB_ON);//调用驱动程序中的 ioctrl 接口，把命令
VIB_ON 传下去，实现硬件操作
                break;
                }
        case VIB_OFF:
                {
                ioctl(fd,VIB_OFF);//调用驱动程序中的 ioctrl 接口，把命令
VIB_OFF 传下去，实现硬件操作
                break;
                }
        default:break;
       }
       return 1;
 }
```

3、相同目录下的新建 Android.mk 如下

Android.mk 文件

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE    := vib-jni
LOCAL_SRC_FILES := vib-jni.c
LOCAL_CFLAGS    := -Werror
LOCAL_LDLIBS    := -llog -lGLESv2 //__android_log_print 函数
include $(BUILD_SHARED_LIBRARY)
```

可以看到，主要是修改 LOCAL_SRC_FILES 指向源文件的名称！

还有一点很重要，如果要使用调试 LOG 打印，也就是__android_log_print 函数。要在 LOCAL_LDLIBS 中添加-llog,如上面的 Android.mk 所示。

4.3 编译JNI模块

```
#cd /home/workspace/android-ndk-r4b/sources/LEDSJNI
```

进到刚才写的 JNI 目录

```
#ndk-build
```

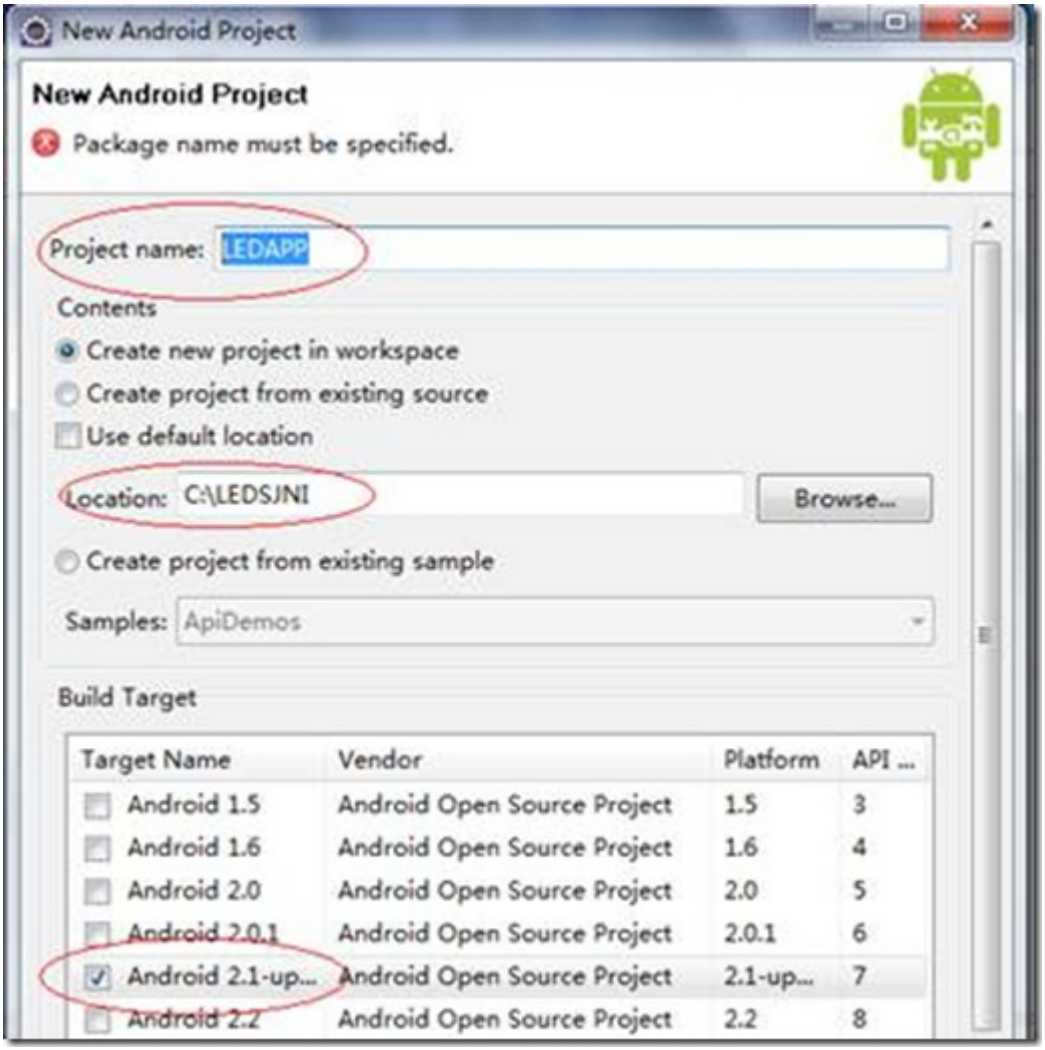
编译 JNI,编译成功后，会在 LEDSJNI 文件夹下生成 libs 和 obj 两个文件夹，并在 LEDSJNI/libs/armeabi 下得到目标文件 libvib-jni.so

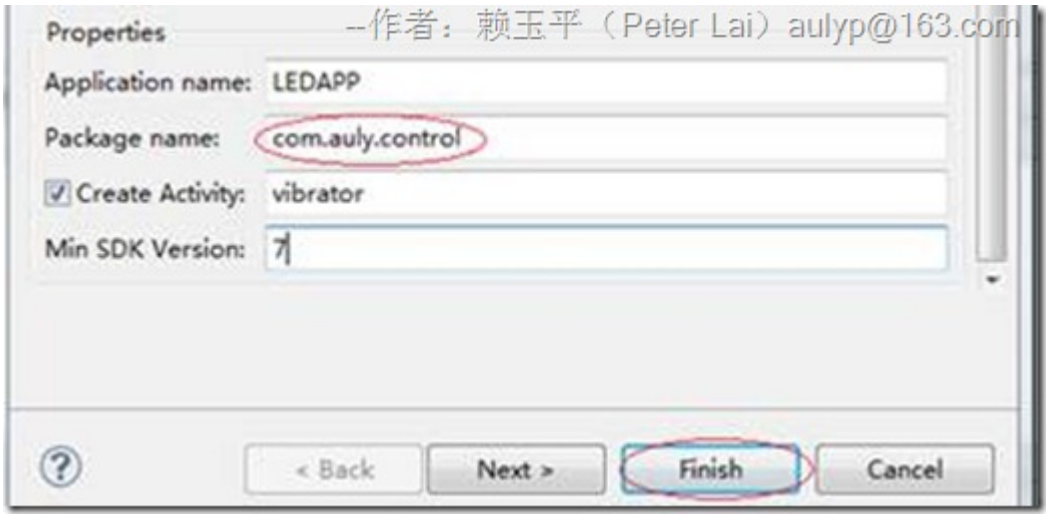
（目前 LEDSJNI 文件夹只有 3 个目录 jni,libs,obj）

4.4 编写JAVA程序

1、Eclipse 新建工程

拷贝 LEDSJNI 目录到 Windows 下，例如 C 盘下。然后在它里面新建 Eclipse 工程。





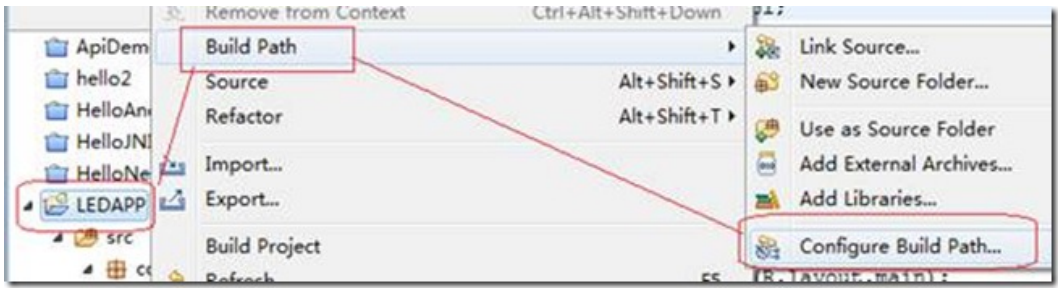
新键工程后，

如果出现如下错误：

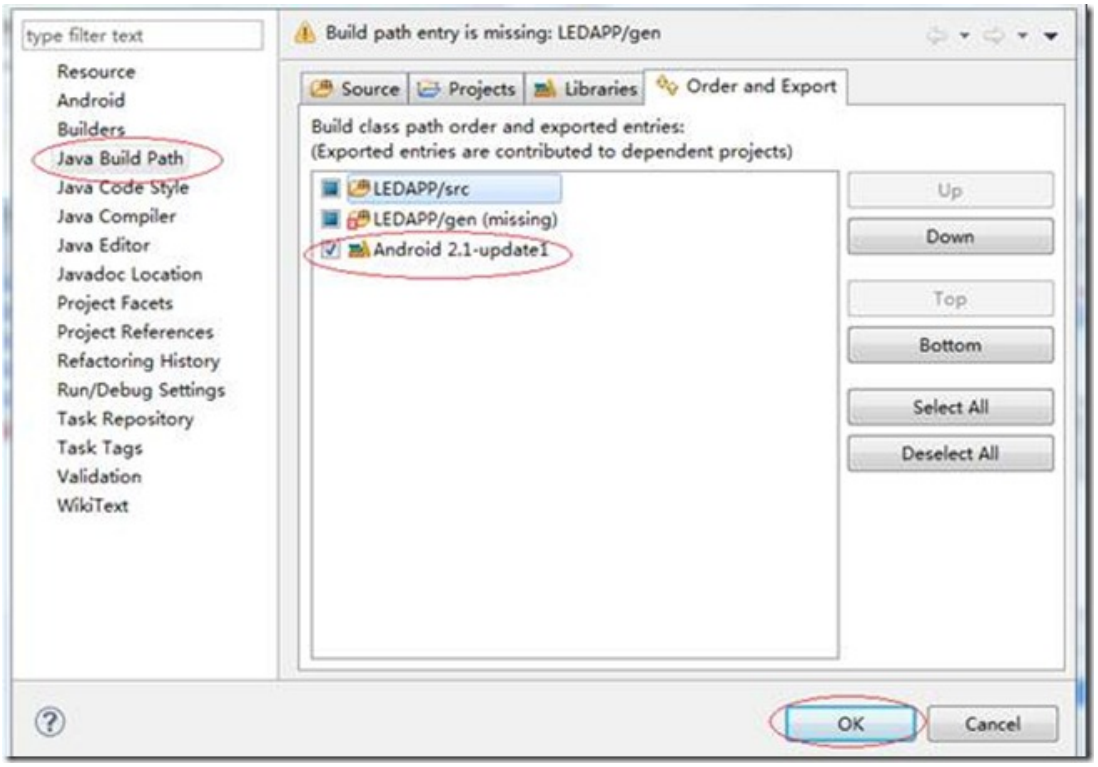
ERROR: Unable to open class file C:/LEDSJNI/gen/com/auly/control/R.java: No such file or directory

解决方法如下：

对着该工程鼠标右键 》bulid path 》configure build path》java build path》order and Export



把里面的 android 2.1 勾上，Build project，就 OK 了



然后 Run as > Android application，就会出现 android 的模拟器了，里面跑个 helloworld 出来。

2、加入 button 和文本输出

程序到上面为止代码是 ADT 自动生成的，似乎与我们一点关系也没有。那我们来改一下代码，因为我们调用 JNI 接口是为了访问驱动程序操作硬件的，例如写，读，打开 LED，关闭 LED 等等，由按钮触发的动作。

第一步是，增加两个 Button，，在 main.xml 里描述一下：

打开 Res > layout> main.xml 文件

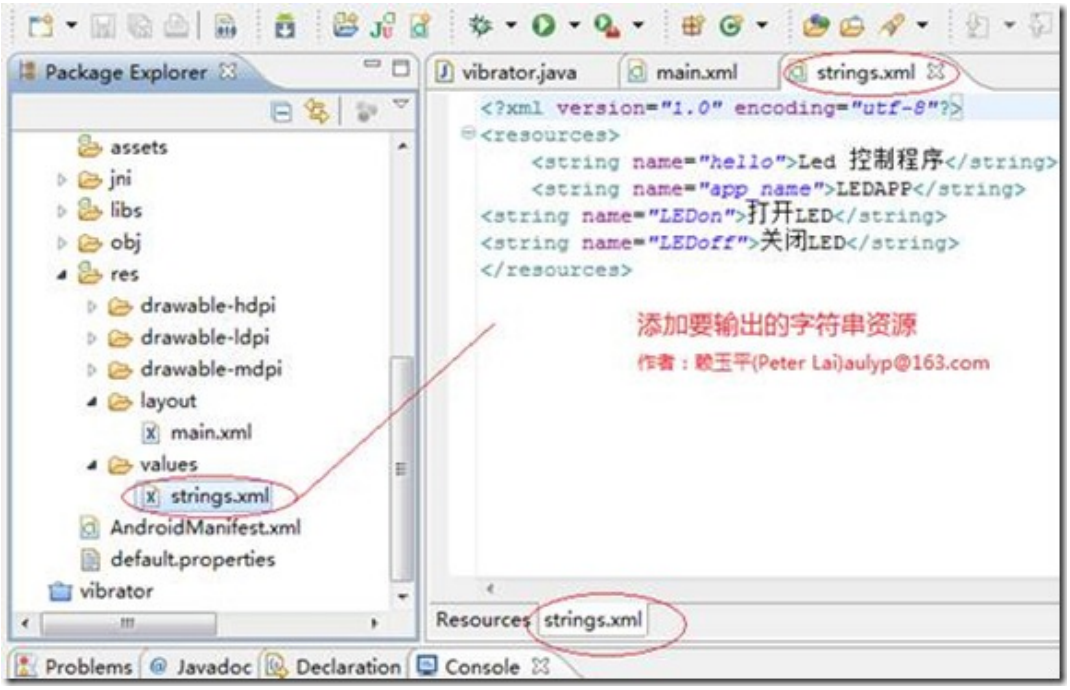
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android: rientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
```

```
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
 <Button android:id="@+id/led_on"/*这表示需要一个唯一的 UID 来作为 Button 的
ID，它的引用名是 led_on。*/
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/LEDon">/*表示这个按钮的文本是来源于另一个资源
描述文件 strings.xml 里的文:字资源 LEDon */
        <requestFocus/>
</Button>
 <Button android:id="@+id/led_off"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/LEDoff">
        <requestFocus/>
 </Button>
</LinearLayout>
```

实际代码中，把注释去掉，否则编译不过的。

3、加入输出字符串资源

工程 > values > strings.xml 文件



修改如下

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
```

```
    <string name="hello">Led 控制程序</string>
    <string name="app_name">LEDAPP</string>
<string name="LEDon">打开 LED</string>
<string name="LEDoff">关闭 LED</string>
</resources>
```

上面的"打开 LED"等资源，就是用在按钮上显示出来的字符串

经过上面的修改，现在程序界面上，已经有如下效果了

鼠标右键工程名>Run as > Android application 运行程序。

4、加入按钮对应的动作

"打开 LED"按扭：调用 JNI 的 IOCTLVIB(VIB_ON);

"关闭 LED"按钮：调用 JNI 的 IOCTLVIB(VIB_OFF);

操作：

在 LEDAPP > src > com.auly.control > **vibrator.java** 文件

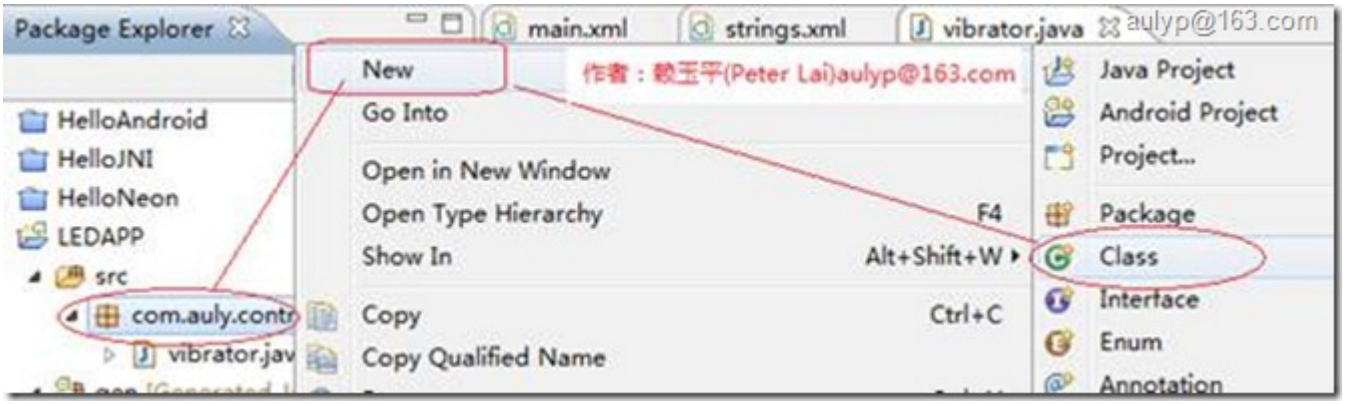
```
package com.auly.control;
/**定义头文件*/
import android.widget.TextView;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.util.Log;
import android.app.Activity;
import android.os.Bundle;
public class vibrator extends Activity {
    /** 定义变量 */
```

```
    public  static  final  int  VIB_ON = 0x11;
    public  static  final  int  VIB_OFF = 0x22;
    vibClass mvibClass;/**定义类*/
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
 /**----------------初始化------------ */
          mvibClass = new vibClass();/*声明类*/
          mvibClass.Init(); //调用 JNI 库里的初始化函数
 /**----------------按钮：打开 LED------------ */
     Button btn1 = (Button)findViewById(R.id.led_on);/*这里用到的 ID，就是
在 main.xml 里定义的 led_on*/
     btn1.setOnClickListener(new View.OnClickListener()
          {
              public void onClick(View v) /**当按钮按下*/
              {
                  mvibClass.IOCTLVIB(VIB_ON);
              }
          });
/**----------------按钮：关闭 LED------------ */
          Button btn2 = (Button)findViewById(R.id.led_off);/*声明按钮,id
main.xml 里有定义*/
          btn2.setOnClickListener(new View.OnClickListener()
          {
              public void onClick(View v) /**当按钮按下*/
              {
                  mvibClass.IOCTLVIB(VIB_OFF);
              }
          });
    }
}
```

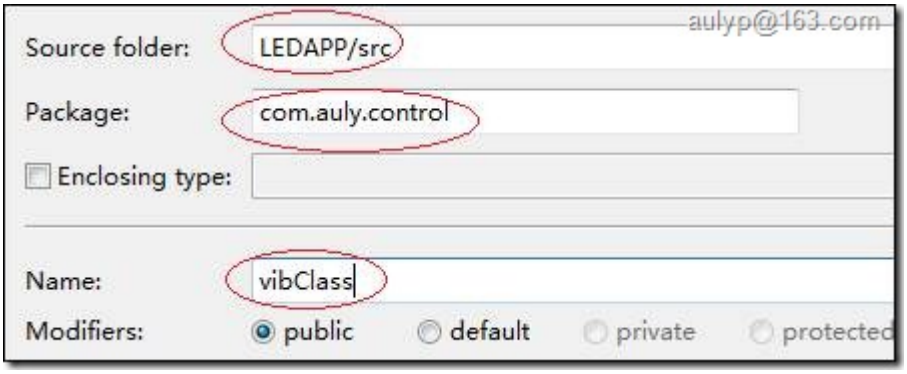
如果在保存时遇到说 save problems，无法保存，请先复制上面的代码，然后，关闭 vibrator.java，提示保存时选不保存，然后在左边资源窗中再次双击打开该文件，把几日才复制下来的内容，粘贴上去，一般就能正常保存，不知道是不是 Eclipse 的不稳定造成的。

5、添加类 vibClass

鼠标右键 com.auly.control > new > Class



填参数：



Finish 后，在/src/com/auly/control/得到如下的类文件

vibClass.java

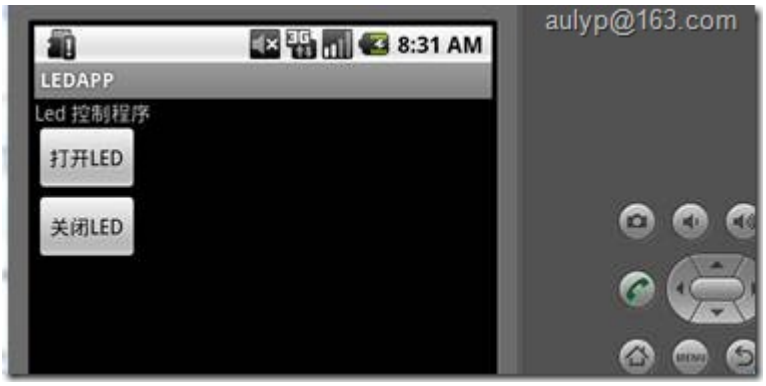
修改如下

```
package com.auly.control;
/*Class for Vibrator --peter*/
public class vibClass {
    static {
        System.loadLibrary("vib-jni");/*加载 JNI 库*/
    }
    /*声明 函数*/
    public static native String stringFromJNI();/*输出字符串
    对应于 JNI 里面的
    jstring Java_com_auly_control_vibClass_stringFromJNI( JNIEnv*
env,jobject thiz )
    */
    public static native int Init();/*初始化函数，对应于 JNI 里面的
                                jint
Java_com_auly_control_vibClass_Init( JNIEnv*  env )
                                */
    public static native int IOCTLVIB(int controlcode);
                                /*IO CTRL 接口
```

```
                                    * 对应于 JNI 里的
 jint Java_com_auly_control_vibClass_IOCTLVIB( JNIEnv*  env, jobject
thiz, jint controlcode )
 */
}
```

4.5 编译运行

鼠标右键工程名，弹出菜单，选择 Run as > Android Application 就可以看到编译过程，编译完成后，会自动调用 android 模拟器，看到如下效果



安装到开发板：

在 C:/LEDSJNI 目录下，会看到 bin 文件夹，里面的 LEDAPP.apk 就是这个程序的安装文件，可以把它安装的开发板上，运行本程序，看控制开发板上的 LED 灯的效果。

步骤：

1、开发板上跑的 kenel 就已经把了 LED 驱动编译在里面了，

可以参考《android 驱动例子(LED 灯控制)》

2、开发板 android 跑起来后，PC 机打开串口工具例如 DNW，打开与开发板连接的 COM 口，然后敲打回车，就会在终端里看到“#”并有光标，表面进入了开发板的命令行终端，

输入命令：

```
#chmod 777 /dev/vib
```

这是为了使得 vib 这个节点可以被我们写的 JNI 操作，不然会 open 失败的，因为 APK 安装的 JNI 模块，权限不够，这个节点是我们的 LED 驱动生成的控制节点。

也可以在 android 文件系统 yaffs 编译时，通过 init.rc 文件来实现这个操作，就是在该文件里随便一行，写上面的命令行，启动时会自动执行！这样就不用手动的改变该节点的属性了。

3、拷贝 LEDAPP.apk 到开发板上，通过安装工具把它安装到开发板上，如果不会安，可以 GOOGLE 一下，

4、运行程序，就能按程序上的近钮来控制开发板上的 LED 亮灭了！

第5章 Ubuntu下进行Android应用开发

针对某些童鞋需要直接在装 Ubuntu 的电脑里开发 Android 的应用程序，因为很多 linux 转过来的 Android 开发人员，已经很熟悉 ubuntu，在电脑上只装有 ubuntu，像我们风火轮团队里的一些高手也常是这么干的，这样可以不用装 VM 虚拟机，ubuntu 独占 PC 资源，编译和编辑更流畅，用得更爽一点。

本教程就是针对这样的需求而设的。

基于 linux 系统作为开发环境！

5.1 Linux开发环境搭建

5.1.1 安装 JDK

首先到oracle的官网上下载linux版本的JDK,

网址为:

http://www.oracle.com/technetwork/java/javase/downloads/jdk-6u26-download-400750.html

我选择的是Linux x86 -Self Extracting Installer，大约81.2M。

下载回来之后，当然是直接运行安装即可.先把这个文件cp到你的安装目录下，然后在你的安装目录直接执行：

```
./jdk-6u26-linux-i586.bin
```

(当然如果执行失败是因为没有执行权限，那么还需要加上执行权限：chmod +x ./jdk-6u26-linux-i586.bin)

安装程序会自动给你安装到当前目录的jdk1.6.0_26目录下。

程序安装好了，接着需要设置环境变量。

执行：

```
sudo gedit /etc/profile
```

在配置文件中加入如下部分：

```
export JAVA_HOME=/home/smartfly/jdk1.6.0_26
export JRE_HOME=/home/ smartfly /jdk1.6.0_26/jre
export PATH=/home/ smartfly /jdk1.6.0_26/bin:$PATH
export CLASSPATH=.:/home/ smartfly /jdk1.6.0_26/lib:/home/ smartfly /jdk1.6.0_26/jre/lib
```

保存，然后注销当前用户，重新登录，环境变量就生效了。

验证是否安装成功：

执行：

```
java -version
```

如果出现:

```
java version "1.6.0_26"Java(TM) SE Runtime Environment (build 1.6.0_26-b06)Java HotSpot(TM) Server VM (build 20.0-b11, mixed mode)
```

这样的信息，就说明 OK 了。

注意，ubuntu 默认已经安装了 openjdk，所以你就算不安装 jdk，执行 java -version 也

不会报错，但是报告的信息和安装了 jdk 是不一样的，请注意区分。

5.1.2 安装eclipse

目前官方推荐开发环境是eclipse+ADT。当然你完全可以采用其他任何一种开发环境，但是我还是选择了官方的推荐，因为一切都是免费的。
下载eclipse的地址是：http://www.eclipse.org/downloads/
我下载的是 32bit linux (可根据自己系统决定用哪个版本)：
Eclipse IDE for Java Developers, 98 MB，
这是一个javaSE开发的基本包，推荐下载classic版本，注意最新ADT 要求eclipse必须是3.5版本以上才可以。
文件名字：eclipse-java-helios-SR1-linux-gtk.tar.gz
当然首先是解压，执行命令：

```
tar zvxf eclipse-java-helios-SR1-linux-gtk.tar.gz
```

解压成功之后，就出现了一个eclipse目录。这个目录就是安装好的eclipse了，进入这个目录，执行eclipse文件，即可启动eclipse。

你可以进去编写一个简单的程序，编译执行一下，来验证下eclispe是否工作正常，你会发现eclipse在linux下的编译执行速度都比windows上要快一点。

当然你也可以自己建立桌面启动快捷方式，便于使用。

5.1.3 下载安装Android SDK Starter Package

Android SDK Starter Package 这个只是 SDK 的核心基本工具，有了它，再利用它的管理工具下载其他你需要到部分。看到这个目录中有个 android 脚本文件了吗？执行它，就会启动 Android SDK and AVD Manager,这个工具可以帮你下载其他你需要的部分，还可以创建模拟机。
下载地址是：http://developer.android.com/sdk/index.html
下载下来到文件是：android-sdk_r11-linux_x86.tgz，大约 26M。
接下来当然是解包了：tar zvxf android-sdk_r11-linux_x86.tgz
解包完毕，就会在当前目录下出现 android-sdk-linux_x86 目录了。这个目录下就是 sdk 的基本工具了。记住这个目录，因为以后你运行工具或者设置 ADT 的时候，会需要这个目录的！

5.1.4 安装ADT（Android Development Tools）插件

ADT插件是google官方开发的Eclipse的一个插件，专门为Android应用开发而准备的。

☐ 启动Eclipse，然后选择**Help> Install New Software...**.

☐ 单击右上角的Add按钮

☐ 在弹出的Add Repository 对话框中，Name一栏目填写"ADT Plugin",在*Location*一栏就有两个选择，一个是填入下载网址：https://dl-ssl.google.com/android/eclipse/，这样就可以网上下载安装。第二个选择是如果我已经下载了ADT Plugin，那么就单击旁边

的Archive按钮，直接找到插件所在到包即可，比如我用的就是第二种，我已经下载下来了，文件名是ADT-11.0.0.zip,5.3M。不管那一种，最后选择OK，即可。

□ 然后在下面的软件列表中，把Developer SDK Tools前面的checkbox选上，然后单击下面的next按钮，然后就等着安装进度条完成吧，其中会出现安装软件的列表，你单击Next即可，然后会出现license,你必须选择agree才行，然后单击finish就完成了（当然其中有个下载过程，可能需要花费一点时间)。安装完成之后，重启eclipse即可。

5.1.5 配置ADT 插件

配置的内容就是要修改在eclipse中到ADT插件的引用指向，需要指向前面安装好的Android SDK目录，还记得这个目录吗？

启动eclipse,选择**Window> Preferences..**

在弹出的**Preferences**对话框中左边选择Android标签，右边SDK location选择你当初Android SDK的目录即可。

比如我到目录是：/home/jackwong/android-sdk-linux_x86，然后单击apply,再单击ok即可。

在这个过程中会弹出一个对话框问你是否要参加google的满意度调查报告。你可以选择参加还是不参加。单击process按钮即可。

安装好之后，你可以常用 **Help> Check for Updates** 选项来保持 ADT 保持最新。

5.1.6 添加Android操作系统和部件

到目前为止，基本配置都差不多了，但是为了开发android应用，还必须添加一个android操作系统，用于模拟器模拟你的应用程序。当然还有一些其他部件需要安装。其实就是通过SDK 工具包中的Android SDK and AVD Manager来完成。当然在eclipse中可以直接启动这个程序：**Window > Android SDK and AVD Manager**.

当然也可以直接运行sdk目录下的tools目录下的android脚本来启动。

推荐的一般是：

SDK Tools，这个你安装了sdk starter package实际上已经装好了。

SDK Platform-tools，这个要装。

SDK platform，这个至少需要一个。也就是操作系统，你可以下载多个，比如2.2，3.1你可以都下载，但是至少必须一个，因为模拟器运行的时候是必须要的。所以，如果你想各种平台都模拟测试一下，那么你就需要各种版本的android都下载一个。

我下载了SDK Platform-tools 5和Android 3.1，Android Compatibility Package三个包。

下载完成之后，Android 的开发环境就搭建好了。

5.2 Hello - android测试程序

验证成果的时候到了，

启动 eclipse,选择 File->New->Project...

然后选择 Android 下的 Android project， 然后填写下面的信息：

project name:HelloAndroid 这是项目名称,也就是包含文件的目录名称

Application name:Hello,Android 这是应用程序名字,也是程序运行的时候显示在屏幕上方的文字

Package Name：my.android 这个不用解释，学过 java 的都应该知道，这是类的包名

Create Activity：helloandroidactivity 这是 activity 的名字，android 程序要显示都是以 activity 来启动的，也就是一个 activity 子类的名字

其他可以不用改动，就用默认值，然后单击 Finish 就完成了项目的创建。

进入项目，打开 helloandroidactivity.java 文件，

把里面的 onCreate 方法的代码更新如下：

```
import android.widget.TextView;
public void onCreate(Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    TextView tv = new TextView(this);
    tv.setText("Hello, Android!=风火轮-树莓派的安卓！");
    setContentView(tv);
}
```

保存，准备运行。

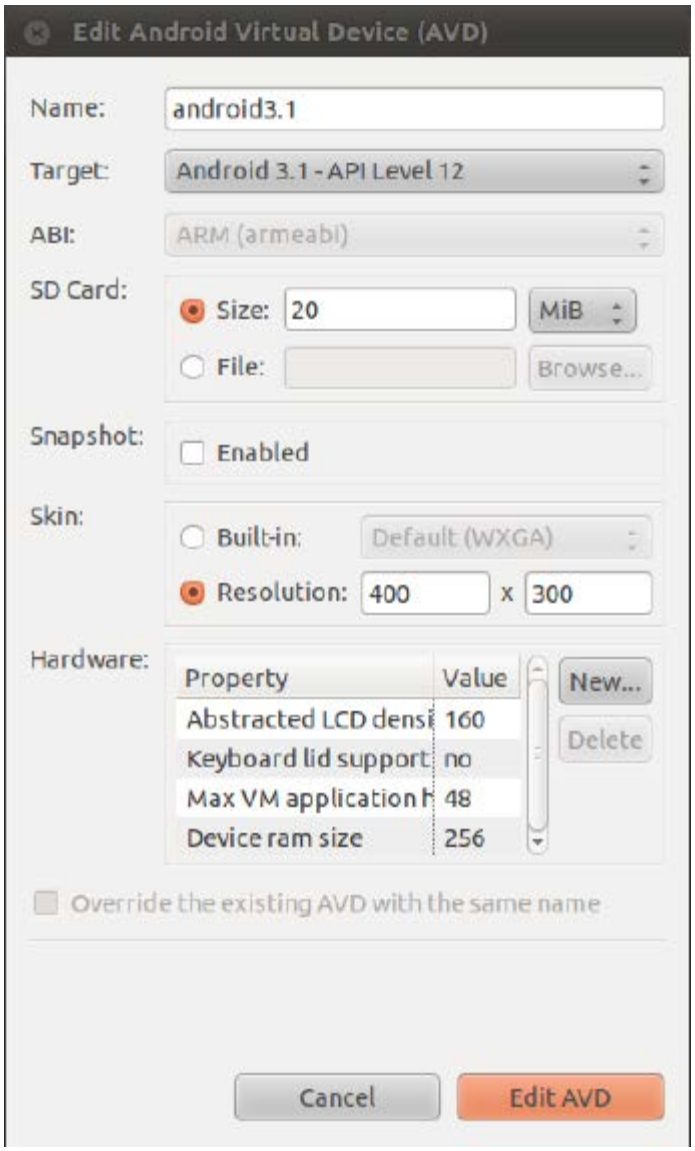
但是如果你就这么运行，我可以担保你很可能会失败。因为android 3.1 确实太费资源了。

你可能会看到模拟器显示一个android标志之后，就停滞不前了。

所以，还需要对模拟器做一下设置。

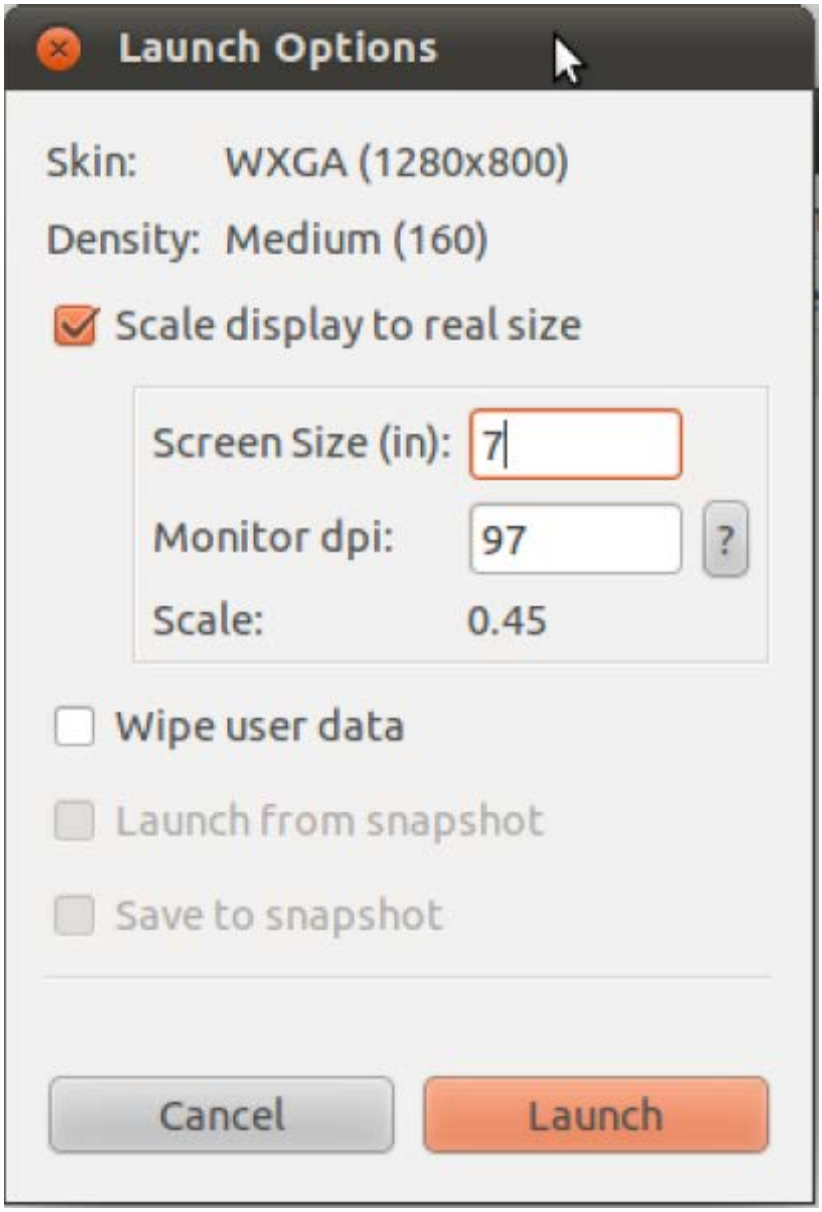
选择windows->Android SDK and AVD Manager

然后单击 new 按钮创建一个新的虚拟机，如下图所示：



其中有个关键点，就是SKin部分，默认是选择WXGA的，分辨率是1280*800，这个分辨率，我的笔记本，反正是没法把模拟机启动成功的。

所以，还是建议你修改一下，采用自定义分辨率吧，比如我用的 480*300 或者 640*400 效果都可以接受。当然还有一种方法是在模拟器的 start...窗口选择：Scale display to real size 也可以，当然默认 3 英寸太小了，调节成 7 英寸或者 5 英寸就效果不错了，但是这个模式速度比较慢，而且看不太清楚，不推荐。如下图所示：



好了，一切都可以了，现在可以Run了，当然Run as成android application了，Eclipse会自动启动模拟器，自动安装apk,自动运行的。

效果如下图所示：