* 1. QT插件
     1. Qt Embedded中如何实现自己的keyboard(转载)

最近的项目是基于Qt，既然是嵌入式设备，难免就要在根据自己的平台来实现键盘的驱动部分，当然是属于Qt一层，而不是更底层的字符设备驱动。

这里要讲的方法是在不重新编译Qt库的情况，把我们自己的键盘驱动作为一个Qt插件集成到我们的程序中。当然也可以把我们自己的键盘驱动部分直接编译到Qt库里面。

Qt中提供的插件机制（Plugin），可以使得我们很容易根据自己的硬件实现自己的键盘代码。

具体步骤：

1. 首先创建一个静态插件库（当然也可以是动态库），我们这边库名叫做optkeypad。

a. 创建一个键盘处理类 KeypadHandler， 多继承于QObject 和 QWSKeyboardHandler, 实现一个关键的函数readKpdData(),这个函数很关键，就是用来把我们的硬件值(hardcode)转换为Qt库中定义的键值。

b. 创建一个插件类 KeypadDriverPlugin, 继承于QKbdDriverPlugin, 这个类主要是创建一个我们自己的键盘处理类KeypadHandler，重新实现QKbdDriverPlugin的create（）和keys()函数，这是两个虚函数，我们要在自己的类中重新实现这两个函数。

c. 我们做成一个静态插件类，所以.pro文件如下。

2. 那么如何使用这个插件类呢，

a. 在我们的应用程序中（一般在main.cpp中）添加宏 Q\_IMPORT\_PLUGIN(optkeypad)。optkeypad是库名，注意不包含linux中的前缀lib和扩展名.a。

b. 在应用程序的.pro文件中添加 QTPLUGIN += optkeypad, 并且要在链接中添加这个库 LIBS += -L(路径) -loptkeypad

c. 接下来要为程序添加环境变量 QWS\_KEYBOARD, export QWS\_KEYBOARD=OptimusKpdHandler:/dev/input/event1 （这个很重要）

注意这个OptimusKpdHandler是怎么来的， 我们的插件类KeypadDriverPlugin要根据这个名字来建立一个键盘处理类KeypadHandler,所以如果这个名字不匹配的话，处理类是不会被创建的。

3. 接下来我们的Qt应用程序中的键值，就是KeypadHandler类的readKpdData()处理后的值。

.pro文件

TEMPLATE = lib

CONFIG += static \

plugin \

release

TARGET = optkeypad

HEADERS += keypadhandler.h \

keypaddriverplugin.h

SOURCES += keypadhandler.cpp \

keypaddriverplugin.cpp

QMAKE\_INCDIR += ../ClockCommon

QMAKE\_LIBS += -lClockCommon

keypadDriverPlugin.h文件

class KeypadDriverPlugin : public QKbdDriverPlugin

{

Q\_OBJECT

public:

KeypadDriverPlugin(QObject \*parent = 0);

~KeypadDriverPlugin();

QWSKeyboardHandler \*create(const QString &driver, const QString &device);

QWSKeyboardHandler \*create(const QString &driver);

QStringList keys() const;

};

类的实现文件

KeypadDriverPlugin::KeypadDriverPlugin(QObject \*parent)

{

}

KeypadDriverPlugin::~KeypadDriverPlugin()

{

}

QStringList KeypadDriverPlugin::keys() const

{

return QStringList() << "OptimusKpdHandler";

}

//

// The create() functions are responsible for returning an instance of

// the keypad driver. We do this only if the driver parameter matches our key.

//

QWSKeyboardHandler \*KeypadDriverPlugin::create(const QString &driver, const QString &device)

{

PR("KeypadDriverPlugin::create###################################: %s\n", driver.toLocal8Bit().constData());

if (driver.toLower() == "optimuskpdhandler")

{

PR("Before creating KeypadHandler\n");

return new KeypadHandler(device);

}

return 0;

}

QWSKeyboardHandler \*KeypadDriverPlugin::create(const QString &driver)

{

if (driver.toLower() == "optimuskpdhandler")

{

PR("Before creating KeypadHandler");

return new KeypadHandler();

}

return 0;

}

keypadHandler.h文件

class KeypadHandler : public QObject, public QWSKeyboardHandler

{

Q\_OBJECT

public:

KeypadHandler(const QString &device = QString("/dev/input/event0"));

~KeypadHandler();

private:

QSocketNotifier \*m\_notifier;

int kbdFd;

private slots:

void readKpdData();

};

类的实现文件

struct InputData

{

unsigned int dummy1;

unsigned int dummy2;

unsigned short type;

unsigned short code;

unsigned int value;

unsigned int dummy3;

unsigned int dummy4;

unsigned int dummy5;

unsigned int dummy6;

};

KeypadHandler::KeypadHandler(const QString &device)

{

setObjectName("Optimus Keypad Handler");

this->kbdFd = ::open(device.toLocal8Bit().constData(), O\_RDONLY, 0);

if (kbdFd > 0)

{

PR("%s opened as keyboard input.\n", device.toLocal8Bit().constData());

g\_Log.Debugf(LOG\_DETAIL\_TRACE, 0, L"%s opened as keyboard input.", device.toLocal8Bit().constData());

this->m\_notifier = new QSocketNotifier(kbdFd, QSocketNotifier::Read, this);

connect(this->m\_notifier, SIGNAL(activated(int)), this, SLOT(readKpdData()));

}

else

{

PR("Cannot open %s for keyboard input. (%s)",

device.toLocal8Bit().constData(), strerror(errno));

g\_Log.Errorf(LOG\_DETAIL\_IMPORTANT, 0, L"Cannot open %s for keyboard input. (%s)",

device.toLocal8Bit().constData(), strerror(errno));

return;

}

}

KeypadHandler::~KeypadHandler()

{

if (kbdFd > 0)

::close(kbdFd);

}

// Key function

void KeypadHandler::readKpdData()

{

InputData event;

int n = read(kbdFd, &event, sizeof(InputData));

if (n != sizeof(InputData))

{

PR("key pressed: n=%d\n", n);

g\_Log.Debugf(LOG\_DETAIL\_TRACE, 0, L"key pressed: n=%d", n);

return;

}

PR("key pressed: type=%d, code=0x%x, value=%d, %s\n",

event.type, event.code, event.value, (event.value != 0)? "(Down)" : "(Up)");

g\_Log.Debugf(LOG\_DETAIL\_TRACE, 0, L"key pressed: type=%d, code=%d, value=%d, %s",

event.type, event.code, event.value, (event.value != 0)? "(Down)" : "(Up)");

Qt::KeyboardModifiers modifiers = Qt::NoModifier;

int unicode = 0xffff;

int key\_code = 0;

// 可以根据自己特定的硬件值来设定。

switch (event.code)

{

case 0x2:

key\_code = Qt::Key\_1;

unicode = '1';

break;

case 0x110:

key\_code = Qt::Key\_Context1;

unicode = 0xffff;

break;

case 0x100:

key\_code = Qt::Key\_Back;

unicode = 0xffff;

break;

default:

break;

}

this->processKeyEvent(unicode, key\_code, modifiers, event.value != 0, false);

* + 1. 我的总结（静态插件）

静态QT插件类似于静态库，会编译成.a，像一般静态库一样，静态链接到QT应用程序中。（即，静态QT插件可以像普通静态库一样使用：添加库的头文件，根据插件类实例化插件对象，并调用对应的初始化函数，完成插件调用）

第二种方法就是QT提供的静态插件调用法，调用方法：

1. .pro文件中添加：QT\_PLUGINS += xxx插件
2. Main.c中添加Q\_IMPORT\_PLUGIN(xxx) (需要头文件《#include <qplugin.h>》)
3. 通过Qtcreater添加对应的库和头文件路径（也可以在.pro文件中直接添加，添加方法百度）
   1. QT信号与槽
      1. 利用C++实现自己的QT信号与槽

<http://blog.csdn.net/fuyunzhishang1/article/details/48345381>

需要理解的核心思想：信号与槽的定义，信号与槽的关联以及如何信号的发出。