QT5.5.1 嵌入式平台 鼠标键盘不能热插拔问题解决（二）

上一篇文章已经介绍了qt键鼠热插拔经历的曲折之路，这篇就具体解决热插拔问题。

==========

如上一篇文章所述，在源码中搜索了键鼠热插拔时的调试信息，

qt.qpa.input: Found mouse at "/dev/input/event0"
qt.qpa.input: Found matching devices ("/dev/input/event0")
qt.qpa.input: Adding mouse at "/dev/input/event0"
qt.qpa.input: create mouse handler for "/dev/input/event0" ""
qt.qpa.input: evdevtouch: Using device discovery

在QT_ROOT/qtbase/src源码目录下搜索上述信息，在中断键入【grep -rn "Adding mouse at"】，结果如下

```
rc-5.5.1/qtbase/src# grep -rn "Adding mouse at"
platformsupport/input/evdevmouse/qevdevmousemanager.cpp:148:    qCDebug
(qLcEvdevMouse) << "Adding mouse at" << deviceNode;
```

发现在QT_ROOT/qtbase/src/platformsupport/input/evdevmouse/qevdevmousemanager.cpp文件中存在相应语句。从命名上就可以猜到该文件是用来管理mouse的，后续验证确实是。将相应的函数贴出来：

```
void QEvdevMouseManager::addMouse(const QString &deviceNode)
{
    qCDebug(qLcEvdevMouse) << "Adding mouse at" << deviceNode;
    QEvdevMouseHandler *handler;
    handler = QEvdevMouseHandler::create(deviceNode, m_spec);
    if (handler) {
        connect(handler, SIGNAL(handleMouseEvent(int,int,bool,Qt::MouseButtons)), this, SLOT(handleMouseEvent(int,in
        connect(handler, SIGNAL(handleWheelEvent(int,Qt::Orientation)), this, SLOT(handleWheelEvent(int,Qt::Orientat
        m_mice.insert(deviceNode, handler);
        QInputDeviceManagerPrivate::get(QGuiApplicationPrivate::inputDeviceManager())->setDeviceCount(
            QInputDeviceManager::DeviceTypePointer, m_mice.count());
    } else {
        qWarning("evdevmouse: Failed to open mouse device %s", qPrintable(deviceNode));
    }
}
```

在同一个文件中的类函数中（贴出部分，如下）可以看出该函数是用于接收信号的槽，且信号为deviceDetected

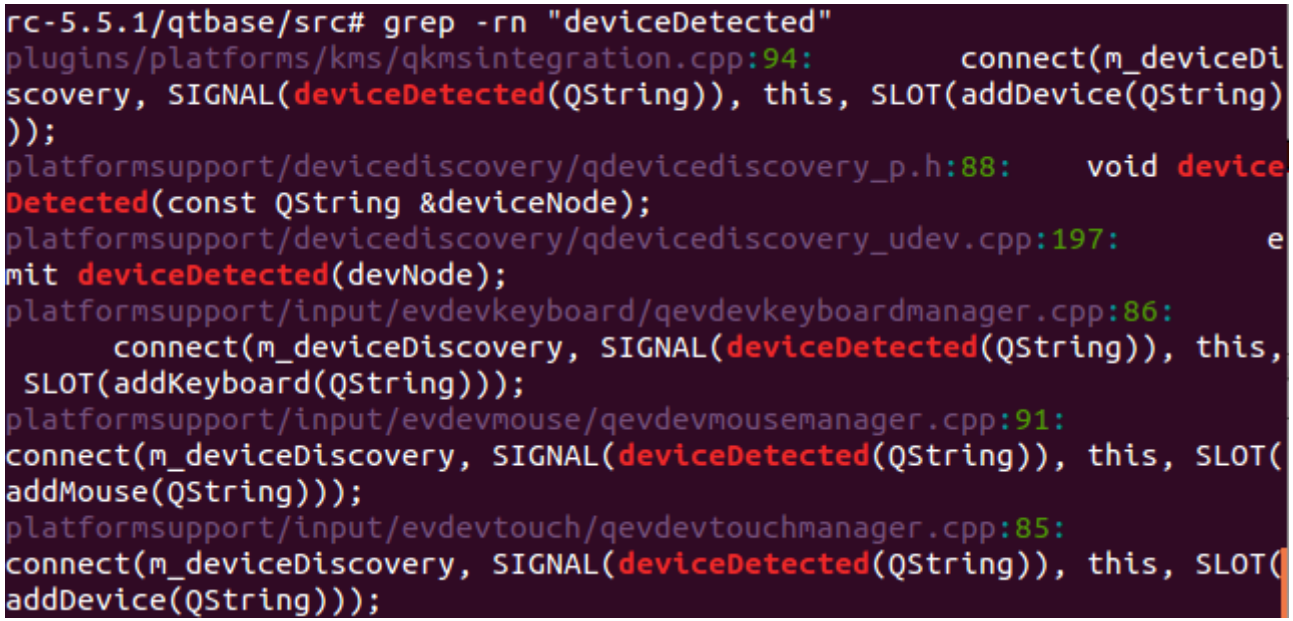
```
。。。。

       qCDebug(qLcEvdevMouse) << "evdevmouse: Using device discovery";
        m_deviceDiscovery = QDeviceDiscovery::create(QDeviceDiscovery::Device_Mouse | QDeviceDiscovery::Device_Touch
        if (m_deviceDiscovery) {
            // scan and add already connected keyboards
            QStringList devices = m_deviceDiscovery->scanConnectedDevices();
            foreach (const QString &device, devices) {
                addMouse(device);
            }
```

```
            connect(m_deviceDiscovery, SIGNAL(deviceDetected(QString)), this, SLOT(addMouse(QString)));
        connect(m_deviceDiscovery, SIGNAL(deviceRemoved(QString)), this, SLOT(removeMouse(QString)));
    }
。。。。
```

同理还存在另外一对信号与槽，全部列举出来：

SIGNAL(deviceDetected(QString)) —— SLOT(addMouse(QString))
SIGNAL(deviceRemoved(QString))—— SLOT(removeMouse(QString))

这两对函数正如猜测的那样，是udev管理热插拔时发出的，具体的发出位置可以搜索【grep -rn "deviceDetected"】，结果如下：



发现在名为qdevicediscovery_udev.cpp的文件中进行的信号发送工作：emit deviceDetected(devNode);

打开qdevicediscovery_udev.cpp所在路径的devicediscovery.pri文件，内容如下：

```
HEADERS += $$PWD/qdevicediscovery_p.h

linux {
    contains(QT_CONFIG, libudev) {
        SOURCES += $$PWD/qdevicediscovery_udev.cpp
        HEADERS += $$PWD/qdevicediscovery_udev_p.h
        INCLUDEPATH += $$QMAKE_INCDIR_LIBUDEV
        LIBS_PRIVATE += $$QMAKE_LIBS_LIBUDEV
    } else: contains(QT_CONFIG, evdev) {
        SOURCES += $$PWD/qdevicediscovery_static.cpp
        HEADERS += $$PWD/qdevicediscovery_static_p.h
    } else {
        SOURCES += $$PWD/qdevicediscovery_dummy.cpp
        HEADERS += $$PWD/qdevicediscovery_dummy_p.h
    }
} else {
    SOURCES += $$PWD/qdevicediscovery_dummy.cpp
}
```

移植过Linux的应该很眼熟，很明显是根据配置单进行选择编译的文件，由于没有配置libudev而默认配置了evdev，因此自动编译的文件名称为 qdevicediscovery_static.cpp/.h。

思路与目标已经明确：

1.在qdevicediscovery_static.cpp文件中实现检测键鼠插入与移除，并发送相应的信号deviceDetected/deviceRemoved即可。
2.检测键鼠的插入与移除可以检测文件系统中/dev/input/目录下设备节点的变化，具体的监听采用QFileSystemWatcher类的路径监听（directoryChanged）。

一、添加监听代码

1.为了方便恢复，复制qdevicediscovery_static.cpp和头文件为qdevicediscovery_hotplug.cpp和qdevicediscovery_hotplug_p.h，修改文件中的static关键字为hotplug（强迫症，可不修改）。

2.由于需要保持监听长时存在，因此监听的实例声明要放在头文件中，并添加相应头文件，由于需要记住上一次添加的设备，因此仿造.cpp文件实现添加一个QStringList变量。

添加头文件：#include <QFileSystemWatcher>

#include <QStringList>

声明监听实例：QFileSystemWatcher *m_fileWatcher;

声明上一次添加设备的列表：QStringList m_devices;
声明信号函数：

```
private slots:
    void handleHotPlugWatch(const QString &path);
```

3.初始化监听变量与信号监听函数实现

```
    // 初始化文件监听器
    m_fileWatcher = new QFileSystemWatcher(this);
    m_fileWatcher->addPath(QString::fromLatin1(QT_EVDEV_DEVICE_PATH));// "dev/input/"
    connect(m_fileWatcher, SIGNAL(directoryChanged(QString)), this, SLOT(handleHotPlugWatch(QString)));
```

信号监听函数实现

```
void QDeviceDiscoveryHotPlug::handleHotPlugWatch(const QString &path)
{
    if(path.compare(QString::fromLatin1(QT_EVDEV_DEVICE_PATH)))
    {
        return;
    }

    QStringList devices;

    // 先移除原来的设备
    foreach (const QString &device, m_devices)
        deviceRemoved(device);

    // 获取现在的设备
    // 注，这里获取的设备已经经过过滤，原因是在对该类进行实例化的时候
    // 已经传进了筛选参数，如：QDeviceDiscovery::Device_Keyboard
    devices = this->scanConnectedDevices();

    // 重新添加设备
    foreach (const QString &device, devices)
        deviceDetected(device);
```

```
}
```

4.修改QStringList QDeviceDiscoveryHotPlug::scanConnectedDevices()函数，保存添加设备的列表

将QStringList devices;改成m_devices.clear();，用以清除上一次的值

将该函数中的devices全部改成m_devices，用以保存新的值。

二、修改devicediscovery.pri文件中的编译选项

如下：

```
。。。
    } else: contains(QT_CONFIG, evdev) {
        SOURCES += $$PWD/qdevicediscovery_hotplug.cpp
        HEADERS += $$PWD/qdevicediscovery_hotplug_p.h
    }
。。。
```

修改文件完成后，重新make/make install，将安装后的qt文件移植到开发板上即可。现在QT5.5.1已经支持热插拔了~~~~

附录：修改后的qdevicediscovery_hotplug.cpp和qdevicediscovery_hotplug_p.h文件如下

1.qdevicediscovery_hotplug.cpp文件具体代码如下：

```
/****************************************************************************
**
** usb input device hot plug function
** by sn02241
**
****************************************************************************/

#include "qdevicediscovery_hotplug_p.h"

#include <QCoreApplication>
#include <QObject>
#include <QHash>
#include <QDir>
#include <QLoggingCategory>
#include <QtCore/private/qcore_unix_p.h>
```

```
#include <linux/input.h>
#include <fcntl.h>

/* android (and perhaps some other linux-derived stuff) don't define everything
 * in linux/input.h, so we'll need to do that ourselves.
 */
#ifndef KEY_CNT
#define KEY_CNT                 (KEY_MAX+1)
#endif
#ifndef REL_CNT
#define REL_CNT                 (REL_MAX+1)
#endif
#ifndef ABS_CNT
#define ABS_CNT                 (ABS_MAX+1)
#endif

#define LONG_BITS (sizeof(long) * 8 )
#define LONG_FIELD_SIZE(bits) ((bits / LONG_BITS) + 1)

static bool testBit(long bit, const long *field)
{
    return (field[bit / LONG_BITS] >> bit % LONG_BITS) & 1;
}

QT_BEGIN_NAMESPACE

Q_LOGGING_CATEGORY(lcDD, "qt.qpa.input")

QDeviceDiscovery *QDeviceDiscovery::create(QDeviceTypes types, QObject *parent)
{
    return new QDeviceDiscoveryHotPlug(types, parent);
}
```

```
QDeviceDiscoveryHotPlug::QDeviceDiscoveryHotPlug(QDeviceTypes types, QObject *parent)
 : QDeviceDiscovery(types, parent),m_fileWatcher(0)    {
    // 初始化文件监听器
    m_fileWatcher = new QFileSystemWatcher(this);
    m_fileWatcher->addPath(QString::fromLatin1(QT_EVDEV_DEVICE_PATH));// "dev/input/"
    connect(m_fileWatcher, SIGNAL(directoryChanged(QString)), this, SLOT(handleHotPlugWatch(QString)));

    qCDebug(lcDD) << "hotplug device discovery for type" << types;
}

QStringList QDeviceDiscoveryHotPlug::scanConnectedDevices()
{
    m_devices.clear();
    QDir dir;
    dir.setFilter(QDir::System);

    // check for input devices
    if (m_types & Device_InputMask) {
        dir.setPath(QString::fromLatin1(QT_EVDEV_DEVICE_PATH));
        foreach (const QString &deviceFile, dir.entryList()) {
            QString absoluteFilePath = dir.absolutePath() + QString::fromLatin1("/") + deviceFile;
            if (checkDeviceType(absoluteFilePath))
                m_devices << absoluteFilePath;
        }
    }

    // check for drm devices
    if (m_types & Device_VideoMask) {
        dir.setPath(QString::fromLatin1(QT_DRM_DEVICE_PATH));
        foreach (const QString &deviceFile, dir.entryList()) {
            QString absoluteFilePath = dir.absolutePath() + QString::fromLatin1("/") + deviceFile;
            if (checkDeviceType(absoluteFilePath))
```

```
                m_devices << absoluteFilePath;
                                                    }
    }

    qCDebug(lcDD) << "Found matching devices" << m_devices;

    return m_devices;
}

bool QDeviceDiscoveryHotPlug::checkDeviceType(const QString &device)
{
    bool ret = false;
    int fd = QT_OPEN(device.toLocal8Bit().constData(), O_RDONLY | O_NDELAY, 0);
    if (!fd) {
        qWarning() << "Device discovery cannot open device" << device;
        return false;
    }

    long bitsKey[LONG_FIELD_SIZE(KEY_CNT)];
    if (ioctl(fd, EVIOCGBIT(EV_KEY, sizeof(bitsKey)), bitsKey) >= 0 ) {
        if (!ret && (m_types & Device_Keyboard)) {
            if (testBit(KEY_Q, bitsKey)) {
                qCDebug(lcDD) << "Found keyboard at" << device;
                ret = true;
            }
        }

        if (!ret && (m_types & Device_Mouse)) {
            long bitsRel[LONG_FIELD_SIZE(REL_CNT)];
            if (ioctl(fd, EVIOCGBIT(EV_REL, sizeof(bitsRel)), bitsRel) >= 0 ) {
                if (testBit(REL_X, bitsRel) && testBit(REL_Y, bitsRel) && testBit(BTN_MOUSE, bitsKey)) {
                    qCDebug(lcDD) << "Found mouse at" << device;
```

```
                ret = true;
            }
        }

        if (!ret && (m_types & (Device_Touchpad | Device_Touchscreen))) {
            long bitsAbs[LONG_FIELD_SIZE(ABS_CNT)];
            if (ioctl(fd, EVIOCGBIT(EV_ABS, sizeof(bitsAbs)), bitsAbs) >= 0 ) {
                if (testBit(ABS_X, bitsAbs) && testBit(ABS_Y, bitsAbs)) {
                    if ((m_types & Device_Touchpad) && testBit(BTN_TOOL_FINGER, bitsKey)) {
                        qCDebug(lcDD) << "Found touchpad at" << device;
                        ret = true;
                    } else if ((m_types & Device_Touchscreen) && testBit(BTN_TOUCH, bitsKey)) {
                        qCDebug(lcDD) << "Found touchscreen at" << device;
                        ret = true;
                    } else if ((m_types & Device_Tablet) && (testBit(BTN_STYLUS, bitsKey) || testBit(BTN_TOOL_PEN,
                        qCDebug(lcDD) << "Found tablet at" << device;
                        ret = true;
                    }
                }
            }
        }

        if (!ret && (m_types & Device_Joystick)) {
            long bitsAbs[LONG_FIELD_SIZE(ABS_CNT)];
            if (ioctl(fd, EVIOCGBIT(EV_ABS, sizeof(bitsAbs)), bitsAbs) >= 0 ) {
                if ((m_types & Device_Joystick)
                    && (testBit(BTN_A, bitsKey) || testBit(BTN_TRIGGER, bitsKey) || testBit(ABS_RX, bitsAbs))) {
                    qCDebug(lcDD) << "Found joystick/gamepad at" << device;
                    ret = true;
                }
            }
        }
```

```
    }

    if (!ret && (m_types & Device_DRM) && device.contains(QString::fromLatin1(QT_DRM_DEVICE_PREFIX)))
        ret = true;

    QT_CLOSE(fd);
    return ret;
}

void QDeviceDiscoveryHotPlug::handleHotPlugWatch(const QString &path)
{
    if(path.compare(QString::fromLatin1(QT_EVDEV_DEVICE_PATH)))
    {
        return;
    }

    QStringList devices;

    // 先移除原来的设备
    foreach (const QString &device, m_devices)
        deviceRemoved(device);

    // 获取现在的设备
    // 注，这里获取的设备已经经过过滤，原因是在对该类进行实例化的时候
    // 已经传进了筛选参数，如：QDeviceDiscovery::Device_Keyboard
    devices = this->scanConnectedDevices();

    // 重新添加设备
    foreach (const QString &device, devices)
        deviceDetected(device);
}

QT_END_NAMESPACE
```

2.qdevicediscovery_hotplug_p.h文件具体代码如下：

```
/*****************************************************************************
**
** usb input device hot plug function
** by sn02241
**
*****************************************************************************/

#ifndef QDEVICEDISCOVERY_HOTPLUG_H
#define QDEVICEDISCOVERY_HOTPLUG_H

//
//  W A R N I N G
//  -------------
//
// This file is not part of the Qt API.  It exists purely as an
// implementation detail.  This header file may change from version to
// version without notice, or even be removed.
//
// We mean it.
//

#include "qdevicediscovery_p.h"
#include <QFileSystemWatcher>
#include <QStringList>
```

```
    QT_BEGIN_NAMESPACE

class QDeviceDiscoveryHotPlug : public QDeviceDiscovery
{
    Q_OBJECT

public:
    QDeviceDiscoveryHotPlug(QDeviceTypes types, QObject *parent = 0);
    QStringList scanConnectedDevices() Q_DECL_OVERRIDE;

private slots:
    void handleHotPlugWatch(const QString &path);

private:
    bool checkDeviceType(const QString &device);

    // 用于检测鼠标键盘热插拔
    QFileSystemWatcher *m_fileWatcher;

    // 原有的设备列表
    QStringList m_devices;
};

QT_END_NAMESPACE

#endif // QDEVICEDISCOVERY_HOTPLUG_H
```