# Qt 之 qInstallMessageHandler（重定向至文件）

# 简述

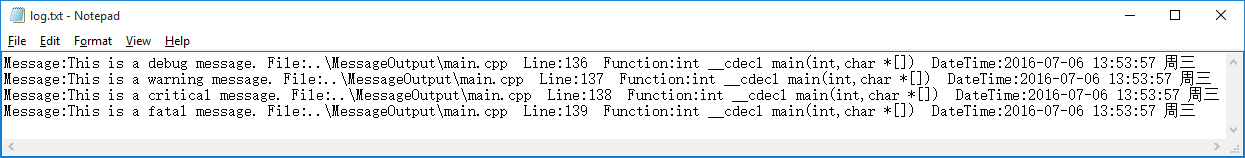
在[Qt之qInstallMessageHandler（输出详细日志）](http://blog.csdn.net/liang19890820/article/details/51838096)一节中，我们讲解了如何使用自定义消息处理程序输出调试信息，而且可以很直观、很方便的得到输出代码所在的文件、函数、行号等信息。

当软件发布的时候，通常都会采用日志输出功能，方便我们后期的跟踪、查找问题。

# 自定义消息处理

下面，我们来根据需要自定义消息处理程序。其中输出格式包含：输出信息、文件、函数、行号、日期时间。这样就很利于我们追踪了。

## 效果



## 源码

void myMessageOutput(QtMsgType type, const QMessageLogContext &context, const QString &msg)

{

*// 加锁*

static QMutex mutex;

mutex.lock();

QByteArray localMsg = msg.toLocal8Bit();

QString strMsg("");

switch(type)

{

case QtDebugMsg:

strMsg = QString("Debug:");

break;

case QtWarningMsg:

strMsg = QString("Warning:");

break;

case QtCriticalMsg:

strMsg = QString("Critical:");

break;

case QtFatalMsg:

strMsg = QString("Fatal:");

break;

}

*// 设置输出信息格式*

QString strDateTime = QDateTime::currentDateTime().toString("yyyy-MM-dd hh:mm:ss ddd");

QString strMessage = QString("Message:%1 File:%2 Line:%3 Function:%4 DateTime:%5")

.arg(localMsg.constData()).arg(context.file).arg(context.line).arg(context.function).arg(strDateTime);

*// 输出信息至文件中（读写、追加形式）*

QFile file("log.txt");

file.open(QIODevice::ReadWrite | QIODevice::Append);

QTextStream stream(&file);

stream << strMessage << "\r\n";

file.flush();

file.close();

*// 解锁*

mutex.unlock();

}

int main(int argc, char \*\*argv)

{

*// 安装消息处理程序*

qInstallMessageHandler(myMessageOutput);

QApplication app(argc, argv);

*// 打印信息*

qDebug("This is a debug message.");

qWarning("This is a warning message.");

qCritical("This is a critical message.");

qFatal("This is a fatal message.");

...

return app.exec();

}

这里，使用QMutex进行加锁、解锁操作，便于我们可以安全的在多线程操作中使用。QFile打开模式为QIODevice::ReadWrite | QIODevice::Append（读写、追加模式），保证了输出信息可以以追加的形式写入文件。