Linux程序开发：QT中的多线程编程（4） PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022_Linux_E7_A8_8B_E5_BA_c103_144937.htm 4、利用定时器机制实现多线程编程 为了避免Qt系统中多线程编程带来的问题，还可以使用系统中提供的定时器机制来实现类似的功能。定时器机制将并发的事件串行化，简化了对并发事件的处理，从而避免了thread-safe方面问题的出现。 在下面的例子中，同时有若干个对象需要接收底层发来的消息（可以通过Socket、FIFO等进程间通信机制），而消息是随机收到的，需要有一个GUI主线程专门负责接收消息。当收到消息时主线程初始化相应对象使之开始处理，同时返回，这样主线程就可以始终更新界面显示并接收外界发来的消息，达到同时对多个对象的控制；另一方面，各个对象在处理完消息后需要通知GUI主线程。对于这个问题，可以利用第3节中的用户自定义事件的方法，在主线程中安装一个事件过滤器，来捕捉从各个对象中发来的自定义事件，然后发出信号调用主线程中的一个槽函数。 另外，也可以利用Qt中的定时器机制实现类似的功能，而又不必担心Thread-safe问题。下面就是有关的代码部分： 在用户定义的Server类中创建和启动了定时器，并利用connect函数将定时器超时与读取设备文件数据相关联： Server:: Server(QWidget *parent) : QWidget(parent) { readTimer = new QTimer(this). //创建并启动定时器 connect(readTimer, SIGNAL(timeout()), this, SLOT(slotReadFile())). //每当定时器超时时调用函数slotReadFile读取文件 readTimer->start(100). } slotReadFile函数负责在定时器超时时，从文件中读取数据，

然后重新启动定时器： int Server::slotReadFile() // 消息读取和处理函数 { readTimer->stop(). //暂时停止定时器计时 ret = read(file, buf ). //读取文件 if(ret == NULL) { readTimer->start(100). //当没有新消息时，重新启动定时器 return(-1). } else 根据buf中的内容将消息分发给各个相应的对象处理…… ； readTimer->start(100). //重新启动定时器 } 在该程序中，利用了类似轮循的方式定时对用户指定的设备文件进行读取，根据读到的数据内容将信息发送到各个相应的对象。用户可以在自己的GUI主线程中创建一个Server类，帮助实现底层的消息接收过程，而本身仍然可以处理诸如界面显示的问题。当各个对象完成处理后，通过重新启动定时器继续进行周期性读取底层设备文件的过程。当然，这种方法适合于各对象对事件的处理时间较短，而底层设备发来消息的频率又相对较慢的情况。 