<http://blog.csdn.net/chenlong12580/article/details/25009095>

Qt中事件传递顺序：

在一个应该程序中，会进入一个事件循环，接受系统产生的事件，并且进行分发，这些都是在exec中进行的。  
下面举例说明：

## 1)首先看看下面一段示例代码：

**[cpp]** [view plain](http://blog.csdn.net/chenlong12580/article/details/25009095) [copy](http://blog.csdn.net/chenlong12580/article/details/25009095)

1. **int** main(**int** argc, **char** \*argv[])
2. {
3. QApplication a(argc, argv);
4. MouseEvent w;
5. w.show();
7. **return** <span style="color:#ff6666;">a.exec();</span>
8. }

## 2)a.exec进入事件循环，调用的是QApplication::exec()；

**[cpp]** [view plain](http://blog.csdn.net/chenlong12580/article/details/25009095) [copy](http://blog.csdn.net/chenlong12580/article/details/25009095)

1. **int** QApplication::exec()
2. {
3. **return** <span style="color:#ff6666;">QGuiApplication::exec();</span>
4. }

## 3)QApplication::exec()调用的是QGuiApplication::exec()；

**[cpp]** [view plain](http://blog.csdn.net/chenlong12580/article/details/25009095) [copy](http://blog.csdn.net/chenlong12580/article/details/25009095)

1. **int** QGuiApplication::exec()
2. {
3. #ifndef QT\_NO\_ACCESSIBILITY
4. QAccessible::setRootObject(qApp);
5. #endif
6. **return** <span style="color:#ff6666;">QCoreApplication::exec();</span>
7. }

## 4)QGuiApplication::exec()调用的是QCoreApplication::exec()；

**[cpp]** [view plain](http://blog.csdn.net/chenlong12580/article/details/25009095) [copy](http://blog.csdn.net/chenlong12580/article/details/25009095)

1. **int** QCoreApplication::exec()
2. {
3. **if** (!QCoreApplicationPrivate::checkInstance("exec"))
4. **return** -1;

7. QThreadData \*threadData = self->d\_func()->threadData;
8. **if** (threadData != QThreadData::current()) {
9. qWarning("%s::exec: Must be called from the main thread", self->metaObject()->className());
10. **return** -1;
11. }
12. **if** (!threadData->eventLoops.isEmpty()) {
13. qWarning("QCoreApplication::exec: The event loop is already running");
14. **return** -1;
15. }

18. threadData->quitNow = **false**;
19. QEventLoop eventLoop;
20. self->d\_func()->in\_exec = **true**;
21. self->d\_func()->aboutToQuitEmitted = **false**;
22. **int** returnCode = <span style="color:#ff6666;">eventLoop.exec();</span>
23. threadData->quitNow = **false**;
24. **if** (self) {
25. self->d\_func()->in\_exec = **false**;
26. **if** (!self->d\_func()->aboutToQuitEmitted)
27. emit self->aboutToQuit(QPrivateSignal());
28. self->d\_func()->aboutToQuitEmitted = **true**;
29. sendPostedEvents(0, QEvent::DeferredDelete);
30. }

33. **return** returnCode;
34. }

## 5)QCoreApplication::exec()调用eventLoop.exec()进行事件循环；

**[cpp]** [view plain](http://blog.csdn.net/chenlong12580/article/details/25009095) [copy](http://blog.csdn.net/chenlong12580/article/details/25009095)

1. **int** QEventLoop::exec(ProcessEventsFlags flags)
2. {
3. Q\_D(QEventLoop);
4. //we need to protect from race condition with QThread::exit
5. QMutexLocker locker(&**static\_cast**<QThreadPrivate \*>(QObjectPrivate::get(d->threadData->**thread**))->mutex);
6. **if** (d->threadData->quitNow)
7. **return** -1;

10. **if** (d->inExec) {
11. qWarning("QEventLoop::exec: instance %p has already called exec()", **this**);
12. **return** -1;
13. }

16. **struct** LoopReference {
17. QEventLoopPrivate \*d;
18. QMutexLocker &locker;

21. **bool** exceptionCaught;
22. LoopReference(QEventLoopPrivate \*d, QMutexLocker &locker) : d(d), locker(locker), exceptionCaught(**true**)
23. {
24. d->inExec = **true**;
25. d->exit = **false**;
26. ++d->threadData->loopLevel;
27. d->threadData->eventLoops.push(d->q\_func());
28. locker.unlock();
29. }

32. ~LoopReference()
33. {
34. **if** (exceptionCaught) {
35. qWarning("Qt has caught an exception thrown from an event handler. Throwing\n"
36. "exceptions from an event handler is not supported in Qt. You must\n"
37. "reimplement QApplication::notify() and catch all exceptions there.\n");
38. }
39. locker.relock();
40. QEventLoop \*eventLoop = d->threadData->eventLoops.pop();
41. Q\_ASSERT\_X(eventLoop == d->q\_func(), "QEventLoop::exec()", "internal error");
42. Q\_UNUSED(eventLoop); // --release warning
43. d->inExec = **false**;
44. --d->threadData->loopLevel;
45. }
46. };
47. LoopReference ref(d, locker);

50. // remove posted quit events when entering a new event loop
51. QCoreApplication \*app = QCoreApplication::instance();
52. **if** (app && app->**thread**() == **thread**())
53. QCoreApplication::removePostedEvents(app, QEvent::Quit);

56. **while** (!d->exit)
57. <span style="color:#ff6666;">processEvents</span>(flags | WaitForMoreEvents | EventLoopExec);

60. ref.exceptionCaught = **false**;
61. **return** d->returnCode;
62. }

## 6)eventLoop.exec()调用QCoreApplication的processEvents进行事件分发；

## 7)调用notify进行分发

QCoreApplication::sendEvent、QCoreApplication::postEvent和QCoreApplication::sendPostedEvents都调用notify进行事件分发；

**[cpp]** [view plain](http://blog.csdn.net/chenlong12580/article/details/25009095) [copy](http://blog.csdn.net/chenlong12580/article/details/25009095)

1. **bool** QCoreApplication::notify(QObject \*receiver, QEvent \*event)
2. {
3. Q\_D(QCoreApplication);
4. // no events are delivered after ~QCoreApplication() has started
5. **if** (QCoreApplicationPrivate::is\_app\_closing)
6. **return** **true**;

9. **if** (receiver == 0) {                        // serious error
10. qWarning("QCoreApplication::notify: Unexpected null receiver");
11. **return** **true**;
12. }

15. #ifndef QT\_NO\_DEBUG
16. d->checkReceiverThread(receiver);
17. #endif

20. **return** receiver->isWidgetType() ? **false** :<span style="color:#ff6666;"> d->notify\_helper</span>(receiver, event);
21. }

## 8)notify调用notify\_helper进行事件分发；

**[cpp]** [view plain](http://blog.csdn.net/chenlong12580/article/details/25009095) [copy](http://blog.csdn.net/chenlong12580/article/details/25009095)

1. **bool** QCoreApplicationPrivate::notify\_helper(QObject \*receiver, QEvent \* event)
2. {
3. // send to all application event filters
4. **if** (<span style="color:#ff6666;">sendThroughApplicationEventFilters</span>(receiver, event))
5. **return** **true**;
6. // send to all receiver event filters
7. **if** (<span style="color:#ff6666;">sendThroughObjectEventFilters</span>(receiver, event))
8. **return** **true**;
9. // deliver the event
10. **return** <span style="color:#ff6666;">receiver->event</span>(event);
11. }

## 9)从上面第8步的代码可以看出事件传递

传递的顺序是：首先传递给全局的事件过滤器，再传递给目标对象的事件过滤器，最终传递给目标对象。