

高级网工实战之旅

60个网络实战案例

¥99
原价153

- 2021薛大龙软考报名入口
- Linux工程师10问10答
- 考华为认证，做高级网工
- 网络安全攻防实战系统班

Qt 事件处理机制（上篇）

在Qt中，事件被封装成一个个对象，所有的事件均继承自抽象类QEvent. 接下来依次谈谈Qt中有谁来产生、分发、接受和处理事件。

作者：佚名 来源：互联网 | 2011-07-01 14:14 [收藏](#) [分享](#)

本篇来介绍Qt 事件处理机制。深入了解事件处理系统对于每个学习Qt人来说非常重要，可以说，Qt是以事件驱动的UI工具集。大家熟知Signals/Slots在多线程的实现也依赖于Qt的事件处理机制。

在Qt中，事件被封装成一个个对象，所有的事件均继承自抽象类QEvent. 接下来依次谈谈Qt中有谁来产生、分发、接受和处理事件：

1、谁来产生事件：最容易想到的是我们的输入设备，比如键盘、鼠标产生的

keyPressEvent, keyReleaseEvent, mousePressEvent, mouseReleaseEvent事件（他们被封装成QMouseEvent和QKeyEvent），这些事件来自于底层的操作系统，它们以异步的形式通知Qt事件处理系统，后文会仔细道来。当然Qt自己也会产生很多事件，比如QObject::startTimer()会触发QTimerEvent. 用户的程序可还以自己定制事件。

2、谁来接受和处理事件：答案是QObject。在Qt的内省机制剖析一文已经介绍QObject类是整个Qt对象模型的心脏，事件处理机制是QObject三大职责（内存管理、内省(intropection)与事件处理制）之一。任何一个想要接受并处理事件的对象均须继承自QObject，可以选择重载QObject::event()函数或事件的处理权转给父类。

3、谁来负责分发事件：对于non-GUI的Qt程序，是由QCoreApplication负责将QEvent分发给QObject的子类-Receiver. 对于Qt GUI程序，由QApplication来负责。

接下来，将通过代码的解析来看看QT是利用event loop从事件队列中获取用户输入事件，又是如何将事件转义成QEvents，并分发给相应的QObject处理。

```
1. #include <QApplication>
2. #include "widget.h"
3. //Section 1
4. int main(int argc, char *argv[])
5. {
6.     QApplication app(argc, argv);
7.     Widget window; // Widget 继承自QWidget
8.     window.show();
9.     return app.exec(); // 进入Qpplication事件循环，见section 2
10. }
11. // Section 2:
12. int QApplication::exec()
13. {
14.     //skip codes
15.     //简单的交给QCoreApplication来处理事件循环=) section 3
16.     return QCoreApplication::exec();
17. }
18. // Section 3
19. int QCoreApplication::exec()
```



编辑推荐

- 头条

沉浸式故事：AR和VR如何改变2021年的市场营销？
- 热点

iOS版微信8.0.3更新了什么 朋友圈和自定义表情突破限制
- 热点

微信一口气更新了 12 个功能
- 聚焦

合理规划：如何为APP选择正确的数据库？
- 关注

为什么手机厂商越来越少做白色前面板的手机呢？

24H热文 一周话题 本月获赞

- iOS 14.5新功能大盘点：个个都是绝技
- 从安卓设备转移数据到iOS的几种方法，掌握...
- iPhone 12价格走势：2个月降10%，5个月...
- 推荐两款iOS端磁力下载工具
- 轻松找到微信接收文件存储位置
- 快捷指令怎么用？玩转iOS14快捷指令全攻略
- 一招搞定手机和电脑的多屏协同
- 合理规划：如何为APP选择正确的数据库？

订阅专栏

+更多



16招轻松掌握PPT技巧
GET职场加薪技能
共16章 | 晒书包
289人订阅学习



```

20. {
21. //得到当前Thread数据
22. QThreadData *threadData = self->d_func()->threadData;
23. if (threadData != QThreadData::current()) {
24.     qWarning("%s::exec: Must be called from the main thread", self->metaObject()->className);
25.     return -1;
26. }
27. //检查event loop是否已经创建
28. if (!threadData->eventLoops.isEmpty()) {
29.     qWarning("QCoreApplication::exec: The event loop is already running");
30.     return -1;
31. }
32. ...
33. QEventLoop eventLoop;
34. self->d_func()->in_exec = true;
35. self->d_func()->aboutToQuitEmitted = false;
36. //委任QEventLoop 处理事件队列循环 ==> Section 4
37. int returnCode = eventLoop.exec();
38. ....
39. }
40. return returnCode;
41. }
42. // Section 4
43. int QEventLoop::exec(ProcessEventsFlags flags)
44. {
45. //这里的实现代码不少，最为重要的是以下几行
46. Q_D(QEventLoop); // 访问QEventloop私有类实例d
47. try {
48. //只要没有遇见exit，循环派发事件
49. while (!d->exit)
50.     processEvents(flags | WaitForMoreEvents | EventLoopExec);
51. } catch (...) {}
52. }
53. // Section 5
54. bool QEventLoop::processEvents(ProcessEventsFlags flags)
55. {
56. Q_D(QEventLoop);
57. if (!d->threadData->eventDispatcher)
58.     return false;
59. if (flags & DeferredDeletion)
60.     QCoreApplication::sendPostedEvents(0, QEvent::DeferredDelete);
61. //将事件派发给与平台相关的QAbstractEventDispatcher子类 =>Section 6
62. return d->threadData->eventDispatcher->processEvents(flags);
63. }
64. #include <QApplication>
65. #include "widget.h"
66. //Section 1
67. int main(int argc, char *argv[])
68. {
69.     QApplication app(argc, argv);
70.     Widget window; // Widget 继承自QWidget
71.     window.show();
72.     return app.exec(); // 进入Qpplication事件循环，见section 2
73. }
74. // Section 2:
75. int QApplication::exec()
76. {
77. //skip codes
78. //简单的交给QCoreApplication来处理事件循环=> section 3
79. return QCoreApplication::exec();
80. }
81. // Section 3
82. int QCoreApplication::exec()
83. {
84. //得到当前Thread数据
85. QThreadData *threadData = self->d_func()->threadData;

```



20个局域网建设改造案例
网络搭建技巧
共20章 | 捷哥CCIE
645人订阅学习



WOT2019全球人工智能技术峰会
通用技术、应用领域、企业赋能三大章节，13大技术专场，60+国内外一线人工智能精英大咖站台，分享人工智能的平台工具、算法模型、语音视觉等技术主题，助力人工智能落地。
共50章 | WOT峰会
0人订阅学习

51CTO学院

新用户免费领VIP月度会员>>



企业云计算架构设计之存储架构设计



Nutanix 超融合基础架构设计指南



VMware vSAN超融合基础架构设计

CTO品牌

+ 更多

CTO训练营第九季招募中

技术经理研习营2021年招募

CTO训练营

申请入营

互联网班

体验营

技术经理

申请加入

能力地图

进化图谱

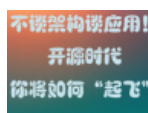
专题推荐

+更多



产品不行技术来撑，看苹果如何打响技术战

产品



不谈架构谈应用！开源时代你将如何“起飞”？

移动



```

86.     if (threadData != QThreadData::current()) {
87.         qWarning("%s::exec: Must be called from the main thread", self->metaObject()->className
88.             return -1;
89.     }
90.     //检查event loop是否已经创建
91.     if (!threadData->eventLoops.isEmpty()) {
92.         qWarning("QCoreApplication::exec: The event loop is already running");
93.         return -1;
94.     }
95.     ...
96.     QEventLoop eventLoop;
97.     self->d_func()->in_exec = true;
98.     self->d_func()->aboutToQuitEmitted = false;
99.     //委任QEventLoop 处理事件队列循环 ==> Section 4
100.    int returnCode = eventLoop.exec();
101.    ....
102.    }
103.    return returnCode;
104. }
105. // Section 4
106. int QEventLoop::exec(ProcessEventsFlags flags)
107. {
108.     //这里的实现代码不少，最为重要的是以下几行
109.     Q_D(QEventLoop); // 访问QEventloop私有类实例d
110.     try {
111.         //只要没有遇见exit，循环派发事件
112.         while (!d->exit)
113.             processEvents(flags | WaitForMoreEvents | EventLoopExec);
114.     } catch (...) {}
115. }
116. // Section 5
117. bool QEventLoop::processEvents(ProcessEventsFlags flags)
118. {
119.     Q_D(QEventLoop);
120.     if (!d->threadData->eventDispatcher)
121.         return false;
122.     if (flags & DeferredDeletion)
123.         QCoreApplication::sendPostedEvents(0, QEvent::DeferredDelete);
124.     //将事件派发给与平台相关的QAbstractEventDispatcher子类 =>Section 6
125.     return d->threadData->eventDispatcher->processEvents(flags);
126. }
127.
128. // Section 6, QTDIR\src\corelib\kernel\qeventdispatcher_win.cpp
129. // 这段代码是完成与windows平台相关的windows c++。以跨平台著称的Qt同时也提供了对Symi
130. // 其事现分别封装在QEventDispatcherSymbian和QEventDispatcherUNIX
131. // QEventDispatcherWin32派生自QAbstractEventDispatcher.
132. bool QEventDispatcherWin32::processEvents(QEventLoop::ProcessEventsFlags flags)
133. {
134.     Q_D(QEventDispatcherWin32);
135.     if (!d->internalHwnd)
136.         createInternalHwnd();
137.     d->interrupt = false;
138.     emit awake();
139.     bool canWait;
140.     bool retVal = false;
141.     bool seenWM_QT_SENDPOSTEDEVENTS = false;
142.     bool needWM_QT_SENDPOSTEDEVENTS = false;
143.     do {
144.         DWORD waitRet = 0;
145.         HANDLE pHandles[MAXIMUM_WAIT_OBJECTS - 1];
146.         QVarLengthArray<MSG> processedTimers;
147.         while (!d->interrupt) {
148.             DWORD nCount = d->winEventNotifierList.count();
149.             Q_ASSERT(nCount < MAXIMUM_WAIT_OBJECTS - 1);
150.             MSG msg;
151.             bool haveMessage;

```



基于React与Vue后，移动开源项目Weex如何定义未来

React



看看16年,移动开发都发生了什么

移动

精选博文 论坛热帖 下载排行

- 企业级Docker镜像仓库Harbor部署与使
- 在最新版proxmox VE 6 部署oracle 19
- Java底层：GC相关
- 为什么大型互联网都需要网关服务？
- Spring Boot 自动配置（auto-configu

读书

+更多



Microsoft SQL Server 2005 技术内幕:T-SQL程序设计

SQL Server 2005微软官方权威参考手册。是Inside Microsoft SQL Server 2005系列书中的第一本，SQL Server类的顶尖之作。全球公认SQL S...

订阅51CTO邮刊

点击这里查看样刊

立即订阅



51CTO服务号



51CTO播客

```

152.     if (!(flags & QEventLoop::ExcludeUserInputEvents) && !d->queuedUserInputEvents.isEmpty()
153.         // process queued user input events
154.         haveMessage = true;
155.         //从处理用户输入队列中取出一条事件
156.         msg = d->queuedUserInputEvents.takeFirst();
157.     } else if (!(flags & QEventLoop::ExcludeSocketNotifiers) && !d->queuedSocketEvents.isEmpty()
158.         // 从处理socket队列中取出一条事件
159.         haveMessage = true;
160.         msg = d->queuedSocketEvents.takeFirst();
161.     } else {
162.         haveMessage = PeekMessage(&msg, 0, 0, 0, PM_REMOVE);
163.         if (haveMessage && (flags & QEventLoop::ExcludeUserInputEvents)
164.             && (msg.message >= WM_KEYFIRST
165.                 && msg.message <= WM_KEYLAST)
166.             || (msg.message >= WM_MOUSEFIRST
167.                 && msg.message <= WM_MOUSELAST)
168.             || msg.message == WM_MOUSEWHEEL
169.             || msg.message == WM_MOUSEHWHEEL
170.             || msg.message == WM_TOUCH
171. #ifndef QT_NO_GESTURES
172.             || msg.message == WM_GESTURE
173.             || msg.message == WM_GESTURENOTIFY
174. #endif
175.             || msg.message == WM_CLOSE)) {
176.             // 用户输入事件入队列，待以后处理
177.             haveMessage = false;
178.             d->queuedUserInputEvents.append(msg);
179.         }
180.         if (haveMessage && (flags & QEventLoop::ExcludeSocketNotifiers)
181.             && (msg.message == WM_QT_SOCKETNOTIFIER && msg.hwnd == d->internalHwnd)
182.             // socket 事件入队列，待以后处理
183.             haveMessage = false;
184.             d->queuedSocketEvents.append(msg);
185.         }
186.     }
187.     ....
188.     if (!filterEvent(&msg)) {
189.         TranslateMessage(&msg);
190.         //将事件打包成message调用Windows API派发出
191.         //分发一个消息给窗口程序。消息被分发到回调函数，将消息传递给windows系统
192.         DispatchMessage(&msg);
193.     }
194. }
195. }
196. } while (canWait);
197. ...
198. return retVal;
199. }
200. // Section 6, QTDIR\src\corelib\kernel\qeventdispatcher_win.cpp
201. // 这段代码是完成与windows平台相关的windows c++。以跨平台著称的Qt同时也提供了对Sym
202. // 其事现分别封装在QEventDispatcherSymbian和QEventDispatcherUNIX
203. // QEventDispatcherWin32派生自QAbstractEventDispatcher.
204. bool QEventDispatcherWin32::processEvents(QEventLoop::ProcessEventsFlags flags)
205. {
206.     Q_D(QEventDispatcherWin32);
207.     if (!d->internalHwnd)
208.         createInternalHwnd();
209.     d->interrupt = false;
210.     emit awake();
211.     bool canWait;
212.     bool retVal = false;
213.     bool seenWM_QT_SENDPOSTEDEVENTS = false;
214.     bool needWM_QT_SENDPOSTEDEVENTS = false;
215.     do {
216.         DWORD waitRet = 0;
217.         HANDLE pHandles[MAXIMUM_WAIT_OBJECTS - 1];

```

```

218. QVarLengthArray<MSG> processedTimers;
219. while (!d->interrupt) {
220.     DWORD nCount = d->winEventNotifierList.count();
221.     Q_ASSERT(nCount < MAXIMUM_WAIT_OBJECTS - 1);
222.     MSG msg;
223.     bool haveMessage;
224.     if (!(flags & QEventLoop::ExcludeUserInputEvents) && !d->queuedUserInputEvents.isEmpty()
225.         // process queued user input events
226.         haveMessage = true;
227.         //从处理用户输入队列中取出一条事件
228.         msg = d->queuedUserInputEvents.takeFirst();
229.     } else if (!(flags & QEventLoop::ExcludeSocketNotifiers) && !d->queuedSocketEvents.isEmpty()
230.         // 从处理socket队列中取出一条事件
231.         haveMessage = true;
232.         msg = d->queuedSocketEvents.takeFirst();
233.     } else {
234.         haveMessage = PeekMessage(&msg, 0, 0, 0, PM_REMOVE);
235.         if (haveMessage && (flags & QEventLoop::ExcludeUserInputEvents)
236.             && (msg.message >= WM_KEYFIRST
237.                 && msg.message <= WM_KEYLAST)
238.             || (msg.message >= WM_MOUSEFIRST
239.                 && msg.message <= WM_MOUSELAST)
240.             || msg.message == WM_MOUSEWHEEL
241.             || msg.message == WM_MOUSEHWHEEL
242.             || msg.message == WM_TOUCH
243.             #ifndef QT_NO_GESTURES
244.             || msg.message == WM_GESTURE
245.             || msg.message == WM_GESTURENOTIFY
246.             #endif
247.             || msg.message == WM_CLOSE)) {
248.             // 用户输入事件入队列，待以后处理
249.             haveMessage = false;
250.             d->queuedUserInputEvents.append(msg);
251.         }
252.         if (haveMessage && (flags & QEventLoop::ExcludeSocketNotifiers)
253.             && (msg.message == WM_QT_SOCKETNOTIFIER && msg.hwnd == d->internalHwnd)
254.             // socket 事件入队列，待以后处理
255.             haveMessage = false;
256.             d->queuedSocketEvents.append(msg);
257.         }
258.     }
259.     ....
260.     if (!filterEvent(&msg)) {
261.         TranslateMessage(&msg);
262.         //将事件打包成message调用Windows API派发出
263.         //分发一个消息给窗口程序。消息被分发到回调函数，将消息传递给windows系统
264.         DispatchMessage(&msg);
265.     }
266. }
267. }
268. } while (canWait);
269. ...
270. return retVal;
271. }
272.
273. // Section 7 windows窗口回调函数 定义在QTDIR\src\gui\kernel\application_win.cpp
274. extern "C" LRESULT QT_WIN_CALLBACK QtWndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam) {
275. {
276. ...
277. //将消息重新封装成QEvent的子类QMouseEvent ==> Section 8
278. result = widget->translateMouseEvent(msg);
279. ...
280. }
281.
282. // Section 7 windows窗口回调函数 定义在QTDIR\src\gui\kernel\application_win.cpp
283. extern "C" LRESULT QT_WIN_CALLBACK QtWndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam) {

```



```

284. {
285. ...
286. //将消息重新封装成QEvent的子类QMouseEvent ==> Section 8
287. result = widget->translateMouseEvent(msg);
288. ...
289. }

```

从Section 1~Section7，Qt进入QApplication的event loop，经过层层委任，最终QEventloop的processEvent将通过与平台相关的QAbstractEventDispatcher的子类QEventDispatcherWin32获得用户的用户输入事件，并将其打包成message后，通过标准Windows API，把消息传递给了Windows OS，Windows OS得到通知后回调QtWndProc，至此事件的分发与处理完成了一半的路程。

小结：Qt 事件处理机制（上篇）的内容介绍完了，在下文中，我们将进一步讨论当我们收到在Windows的回调后，事件又是怎么一步步打包成QEvent并通过QApplication分发给最终事件的接受和处理者QObject::event.请继续看Qt 事件处理机制（下篇）。***希望本文能帮你解决问题！

【编辑推荐】

1. Qt For Symbian截获程序前后台切换事件
2. 初学者文档 QT中窗口刷新事件
3. Qt 多线程之逐线程事件循环 下篇
4. 详解 QT 源码之 Qt 事件机制原理
5. QT源码之Qt信号槽机制与事件机制的联系
6. 详解 Qt 事件过滤器

【责任编辑：李程站 TEL：（010）68476606】

点赞 2

Qt

事件

分享:



大家都在看 猜你喜欢



值得推荐的五款免费网络漏洞扫描器



将在2021年颠覆业务发展的十种技术



Angular、React与Vue，那个框架更好？



今天我才发现Redis有7种数据类型...

好课推荐



企业云计算架构设计之存储架构设计



Nutanix 超融合基础架构设计指南



VMware vSAN超融合基础架构设计



备战2021软考--系统架构设计师视频课程专题

