信号与槽

原创 Qt 学习 Qt 学习 2019-11-16

信号与槽用于两个对象之间的通信。信号与槽机制是 Qt 的核心特征，也是 Qt 不同于其它开发框架的最突出特征。本篇对初学者可以大致浏览查看，往后的学习中对信号与槽有一些更深刻的理解后，再回过来了解信号与槽的相关知识，理解效果会更佳。

本篇目录

1. 概述

Qt 提供了信号与槽机制用于完成界面操作的响应，信号与槽机制是完成任意两个 Qt 对象之间的通信机制。

其中，信号会在某个特定的情况或动作下被触发，槽是等同于接受并处理信号的函数。例如，若要将一个窗口的变化情况通知给另一个窗口部件，则一个窗口部件发送信号，另一个窗口部件的槽接收此信号并进行相应的操作，即可实现两个窗口部件之间的通信。

每个 Qt 对象都包含若干个预定义的信号和若干个定义的槽。某一个特定时间发生时，一个信号被发送，与信号相关联的槽则会响应信号并完成相应的处理。当一个类被继承时，该类的信号和槽也同时被继承，也可以根据需要自定义信号和槽。

2. 信号与槽的关联方式

信号与槽的关联使用的是 `QObject` 类的 `connect()` 函数，该函数原型如下：

```
[static] QMetaObject::Connection QObject::connect(
   const QObject *sender,   // 发射信号的对象
   const char *signal,      // 要发射的信号
   const QObject *receiver, // 接收信号的对象
   const char *method,      // 要执行的槽函数
   Qt::ConnectionType type = Qt::AutoConnection) // 信号与槽的关联类型
```

信号和槽关联类型：

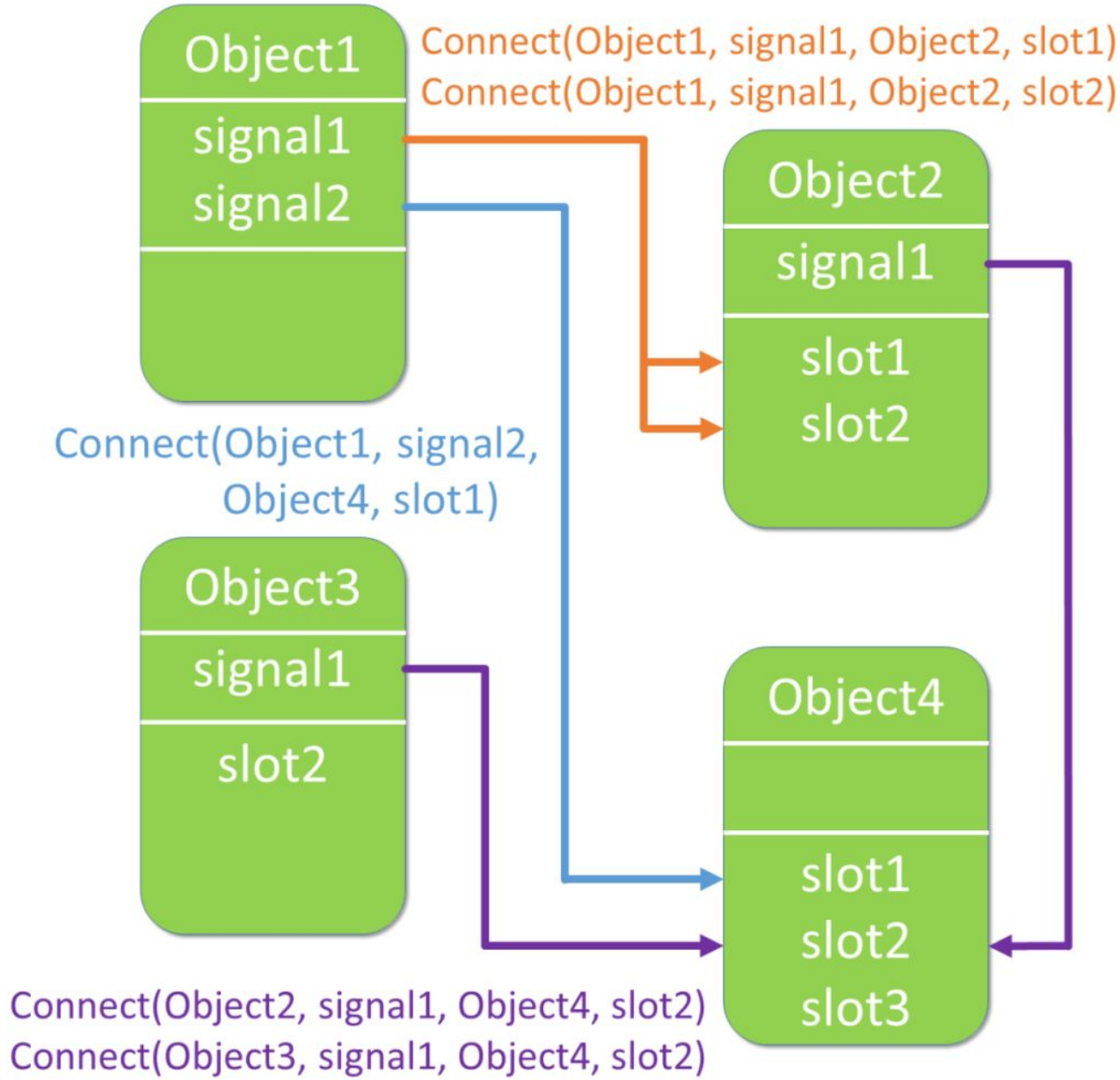
```
Qt::AutoConnection           -> 自动关联，默认值
Qt::DirectConnection         -> 直接关联
Qt::QueuedConnection         -> 队列关联
Qt::BlockingQueuedConnection -> 阻塞队列关联
Qt::UinqueConnection         -> 唯一关联
```

常用的信号与槽的连接方式如下：

```
1  connect(Object1, SIGNAL(signal), Object2, SLOT(slot));
```

其中，`signal` 为对象 `Object1` 的信号，`slot` 为对象 `Object2` 的槽。下图为常见的集中信号与槽的关联方式

- 一个信号可以与另一个信号相连
- 同一个信号可以与多个槽相连
- 同一个槽可以响应多个信号


Object1
signal1
signal2
Connect(Object1, signal1, Object2, slot1)
Connect(Object1, signal1, Object2, slot2)
Object2
signal1
slot1
slot2
Connect(Object1, signal2,
Object4, slot1)
Object3
signal1
slot2
Object4
slot1
slot2
slot3
Connect(Object2, signal1, Object4, slot2)
Connect(Object3, signal1, Object4, slot2)

3. 信号与槽机制的优点

- **类型安全**

需要关联的信号和槽的签名必须是等同的，即信号的参数类型和参数个数与接收该信号的槽的参数类型和参数个数相同。不过，一个槽的参数个数是可以少于信号的参数个数的，但缺少的参数必须是信号参数的最后一个或几个参数。如果信号和槽的签名不符，编译器会报错。

- **松散耦合**

信号与槽机制减弱了 Qt 对象的耦合度。激发信号的 Qt 对象无须知道是哪个对象的哪个槽需要接收它发出的信号，它仅需做的是在适当的时间发送适当的信号就可以了，而不需要关心发出的信号有没有被接收到，更不需要知道哪个对象的哪个槽接收到了信号。同样，对象的槽也不知道是哪些信号关联了自己，而一旦关联信号和槽，Qt 就保证了适合的槽得到了调用。即使关联的对象在运行时被删除，应用程序也不会崩溃。

使用信号与槽应该注意的几点：

- 一个类若要支持信号与槽，就必须从 `QObject` 或 `QObject` 的子类继承
- Qt 信号与槽机制不支持对模板的使用
- 在类声明的最开始处添加 `Q_OBJECT` 宏
- 槽中的参数类型要和信号参数的类型相对应，且不能比信号的参数多；
- 信号只用声明，没有定义，且返回值为 `void` 类型
- 应用在关联函数 `connect()` 中的信号函数和槽函数的参数只能有类型，不能有变量名

- 对于信号和槽，必须使用 `SIGNAL()` 和 `SLOT()` 宏，将其参数转化为 `const char *` 类型
- 信号和槽函数声明时必须分别使用 `signals` 和 `slots` 关键字

4. 信号与槽机制的效率

信号与槽机制增强了对象间通信的灵活性，然而，这也损失了一些性能。同回调函数相比，信号与槽机制运行速度有些慢。通常，通过传递一个信号来调用槽函数会比直接调用非虚函数的运行速度慢 10 倍。主要原因如下：

- 需要定位接收信号的对象
- 安全地遍历所有关联（如一个对象关联多个槽的情况）
- 编组解组传递的函数
- 在多线程时，信号可能需要排队等候

然而，创建堆对象的 `new` 操作及删除堆对象的 `delete` 操作相比，信号与槽的运行代价很小。信号与槽机制导致的这点性能损失对实时应用程序是可以忽略的。同信号与槽提供的灵活性和简便性相比，这点性能损失也是值得的。

5. 信号和槽断开关联

可以通过 `disconnect()` 函数断开信号和槽的关联，其函数原型如下

```
[static] bool QObject::disconnect(
   const QObject *sender,
   const char *signal,
   const QObject *receiver,
   const char *method)
```

使用方法如下：

- 断开一个对象的所有信号的所有关联

```
disconnect(myObject, 0, 0, 0);
myObject->disconnect();        // 与第一行代码等价
```

- 断开一个指定信号的所有关联

```
disconnect(myObject, SIGNAL(mySignal()), 0, 0);
myObject->disconnect(SIGNAL(mySignal()));      // 与第一行代码等价
```

- 断开一个指定接收对象的所有关联

```
disconnect(myObject, 0, myReceiver, 0);
myObject->disconnect(myReceiver);      // 与第一行代码等价
```

- 断开指定信号和槽的关联

```
disconnect(myObject, SIGNAL(mySignal()), myReceiver, SLOT(mySlot()));
myObject->disconnect(SIGNAL(mySignal()), myReceiver, SLOT(mySlot())); // 与第一行代码等价
myObject->disconnect(myConnection); // 与前两行代码等价
                                    // myConnection 是进行关联时 connect() 的返回值
```

5. Qt 元对象系统

Qt 5 元对象系统提供了对象间的通信机制（信号与槽）、运行时类型信息和动态属性系统的支持，是标准 C++ 的一个扩展，它使 Qt 能够更好地实现 GUI 图形用户界面编程。Qt 5 元对象系统不支持 C++ 模板，尽管该模板扩展了标准 C++ 的功能。但是，元

对象系统提供了模板无法提供的一些特性。Qt 5 元对象系统基于以下三个事实：

1. 基类 `QObject`：任何需要使用元对象系统功能的类必须继承自 `QObject`。

2. `Q_OBJECT` 宏：`Q_OBJECT` 宏必须出现在类的私有声明区中，用于启动元对象的特性。

3. 元对象编译器 (Meta-Object Compiler, MOC)：为 `QObject` 子类实现元对象特性提供必须的代码实现。

参考文献

1. Qt 5.12.4 帮助文档
2. 陆文周. Qt 5 开发实例（第4版）2019.
3. 霍亚飞. Qt Creator 快速入门, 2017.

相关阅读：

《代码化 UI 设计》
《可视化 UI 设计（设计器 Qt Designer 实现）》
《UI 文件设计与运行机制》
《编写一个 Hello World 程序》