1. QObject是以对象树的形式组建起来的

创建一个QObject对象时，构造函数会接受一个Parent对象指针，即父对象指针。因此，该对象在创建的时候，父对象就知道了有一个子对象生成了，并将该子对象添加到起父对象的children()列表中。

当父对象析构时，**这个列表中的所有对象也会被析构。**

二．QWidget是能够在屏幕上显示的一切组件的父类。

* + **QWidget继承自QObject，因此也继承了这种对象树关系。**当用户关闭一个对话框的时候，应用程序将其删除，那么，我们希望属于这个对话框的按钮、图标等应该一起被删除。事实就是如此，因为这些都是对话框的子组件。

**我们也可以自己删除子对象，它们会自动从其父对象列表中删除。**

**三．对象树解决了Qt的内存回收问题**

**当对象树中的QObject对象delete时，如果这个对象有Parent对象，那么它会自动从Parent对象的children()列表中删除；如果该对象有子对象，那么自动delete每一个子对象。**

**对象树的析构顺序和构造顺序是相反的，构造时是从父对象到子对象的顺序构造，析构是从子对象到父对象的顺序析构，**

**四．代码分析**

{

QWidget window;

QPushButton quit("Quit", &window);

}

作为父组件的 window 和作为子组件的 quit 都是QObject的子类（事实上，它们都是QWidget的子类，而QWidget是QObject的子类）。这段代码是正确的，quit 的析构函数不会被调用两次，因为标准 C++要求，**局部对象的析构顺序应该按照其创建顺序的相反过程**。因此，这段代码在超出作用域时，会先调用 quit 的析构函数，将其从父对象 window 的子对象列表中删除，然后才会再调用 window 的析构函数。

但是，如果我们使用下面的代码：

{

QPushButton quit("Quit");

QWidget window;

quit.setParent(&window);

}

作为父对象的 window 会首先被析构，因为它是最后一个创建的对象。在析构过程中，它会调用子对象列表中每一个对象的析构函数，也就是说， quit 此时就被析构了。在 window 析构之后，quit 也会被析构，因为 quit 也是一个局部变量，在超出作用域的时候当然也需要析构。但是，这时候已经是第二次调用 quit 的析构函数了，C++ 不允许调用两次析构函数，因此，程序崩溃了。

所以，在 Qt 中，尽量在构造的时候就指定 parent 对象，并且大胆在堆上创建。