1 按钮的创建

```
#include "widget.h"
#include <QPushButton>
Widget::Widget(QWidget *parent)
  : QWidget(parent)
{
  // button = new QPushButton ;
  // button = new QPushButton(this) ;//构造函数时 指定父对象
  button = new QPushButton("登入",this) ;//构造函数时指定父对象和设置文本
  //如果不给按钮指定父对象 那么按钮和窗口是单独显示 如果给按钮指定了父对象,只要父对象显示了,按钮也会显示
   button->show();
   //指定按钮的父类是窗口
   // button->setParent(this);//指定按钮的父亲是窗口
   button->resize(300,200);//设置按钮的大小
   button->move(100,100);//设置按钮在窗口中的位置
   // button->setText("登入");//设置按钮的文本

 }
```

```
#ifndef WIDGET_H
#define WIDGET_H
#include <QPushButton>
#include <QWidget>

class Widget : public QWidget
{
 Q_OBJECT

public:
 Widget(QWidget *parent = 0);
```

```
12  ~Widget();
13  QPushButton *button;
14 };
15
16 #endif // WIDGET_H
17
```

2 对象模型（对象树）

在Qt中创建对象的时候会提供一个Parent对象指针，下面来解释这个parent到底是干什么的。

l QObject是以对象树的形式组织起来的。

n 当你创建一个QObject对象时，会看到QObject的构造函数接收一个QObject指针作为参数，这个参数就是 parent，也就是父对象指针。

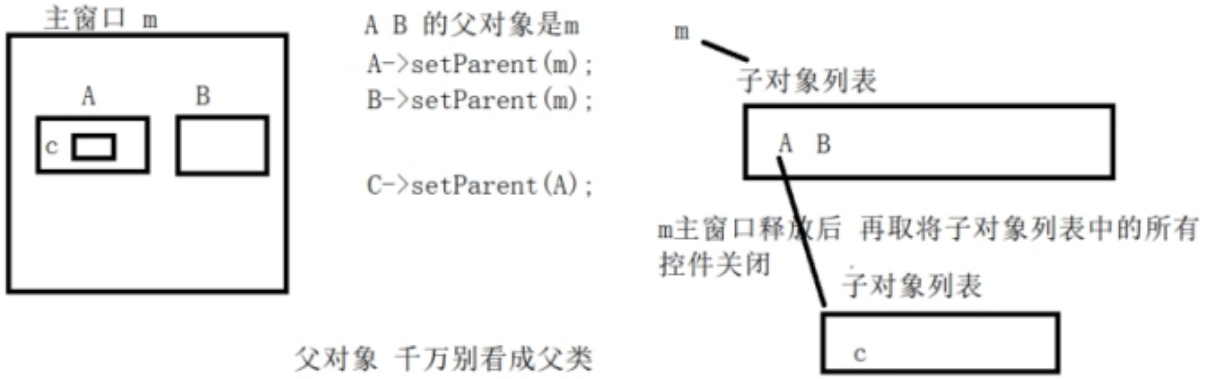
这相当于，**在创建QObject对象时，可以提供一个其父对象，我们创建的这个QObject对象会自动添加到其父对象的children()列表。**

n 当父对象析构的时候，这个列表中的所有对象也会被析构。（注意，这里的父对象并不是继承意义上的父类！）

l QWidget是能够在屏幕上显示的一切组件的父类。

n QWidget继承自QObject，因此也继承了这种对象树关系。一个孩子自动地成为父组件的一个子组件。因此，它会显示在父组件的坐标系统中，被父组件的边界剪裁。例如，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。

n 当然，**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。**比如，当我们删除了一个工具栏时，其所在的主窗口会自动将该工具栏从其子对象列表中删除，并且自动调整屏幕显示。



Qt 引入对象树的概念，在一定程度上解决了内存问题。

当一个QObject对象在堆上创建的时候，Qt 会同时为其创建一个对象树。不过，对象树中对象的顺序是没有定义的。这意味着，销毁这些对象的顺序也是未定义的。

任何对象树中的 QObject对象 delete 的时候，如果这个对象有 parent，则自动将其从 parent 的children()列表中删除；如果有孩子，则自动 delete 每一个孩子。Qt 保证没有QObject会被 delete 两次，这是由析构顺序决定的。

如果QObject在栈上创建，Qt 保持同样的行为。正常情况下，这也不会发生什么问题。

来看下下面的代码片段：

```
{
   QWidget window;
   QPushButton quit   = QPushButton ("退出", &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：

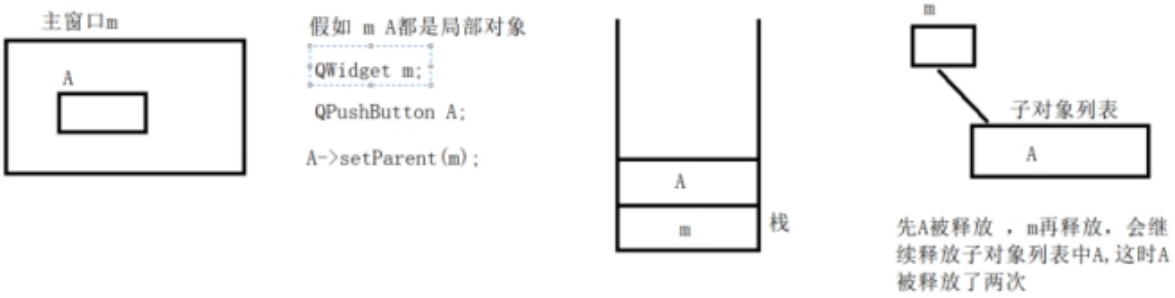
```
{
   QPushButton quit("Quit");
```

```
    QWidget window;
    quit.setParent(&window);
}
```

情况又有所不同，析构顺序就有了问题。我们看到，在上面的代码中，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。然后，代码继续执行，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。

由此我们看到，Qt 的对象树机制虽然帮助我们在一定程度上解决了内存问题，但是也引入了一些值得注意的事情。这些细节在今后的开发过程中很可能时不时跳出来烦扰一下，所以，我们最好从开始就养成良好习惯。

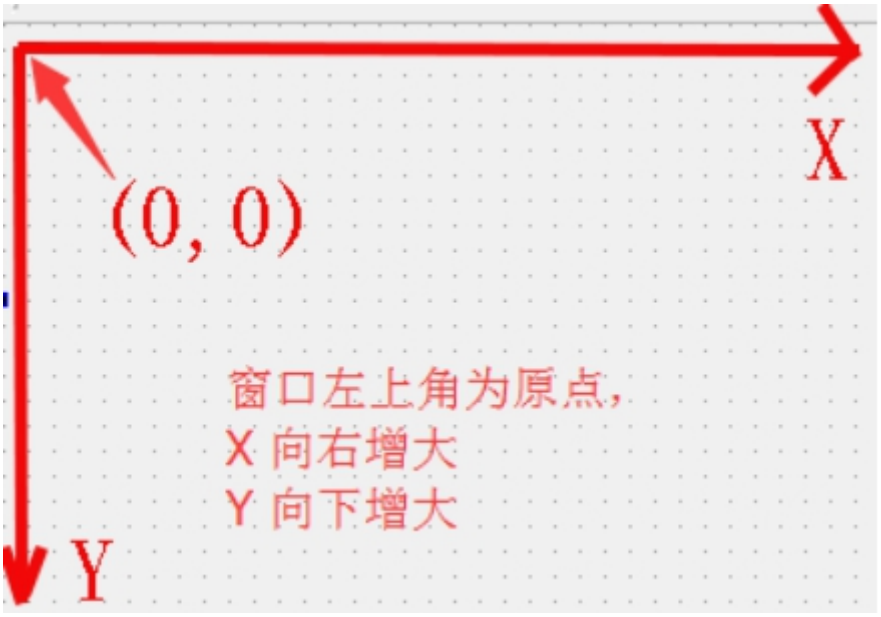
在 Qt 中，尽量在构造的时候就指定 parent 对象，并且大胆在堆上创建。



3 Qt窗口坐标体系

坐标体系：

以左上角为原点（0,0），X向右增加，Y向下增加。



对于嵌套窗口，其坐标是相对于父窗口来说的。