<https://www.cnblogs.com/cute/archive/2011/03/01/1968047.html>

就分配空间和释放空间而言，Qt的处理有点乏味，要不然是通过隐式共享的containers，要不然就是通过QObject的父子关系模式。但总有些时候我们需要在堆上分配一些空间，问题来了，我们该在哪里delete它，如何能够确保不产生内存泄露呢？   
QScopedPointer就为了解决这个问题而生的，哈哈 QScopedPointer在其生命期结束后会自动删除它所指的对象。

1. void foo()
2. {
3. QScopedPointer<int> i(new int(42));
4. …
5. if (someCondition)
6. return; // 我们在堆上构造的整数这时会在这里删除或者是在下面
7. …
8. } // … 也可能在这里

复制代码

这样就可以确保我们在堆上为整数42分配的空间不会产生内存泄露，同时我们也不用手动delete它，哈哈。   
那我们如何访问QScopedPointer 所指的对象呢？QScopedPointer重新实现了operator\* 和operator->，因此我们可以像下面这样使用它：

1. QScopedPointer<int> i(new int(42));
2. \*i = 43;

复制代码

有些运算符是不支持的，如赋值运算符：

1. QScopedPointer<int> i(new int(42));
2. i = new int(43); // 编译不通过
3. i.reset(new int(43)); // 正确

复制代码

operator T\*()也是没有的：

1. int \*foo()
2. {
3. QScopedPointer<int> i(new int(42));
4. …
5. return i; // thankfully, this does not compile.
6. }

复制代码

看到错误没？在上面的代码中一旦我们return，我们构造的对象将被删除因为i的生命期已经结束，我们将会返回一个野指针，这可能会导致崩溃。如果真要返回我们应该像下面这样：

1. int \*foo()
2. {
3. QScopedPointer<int> i(new int(42));
4. …
5. if (someError)
6. return 0; // our integer is deleted here
7. return i.take(); // from now on, our heap object is on its own.
8. }

复制代码

通过调用take()方法，我们告诉QScopedPointer它的工作已经做完，现在由我们来接管在堆上分配对象的所有权，哈哈   
上面的只是针对new而言的，那么如果是malloc或者operator new[]构造数组呢？这里我们需要使用QScopedPointer的第二个参数：

1. QScopedPointer<int, QScopedPointerPodDeleter> pod(static\_cast<int \*>(malloc(sizeof int)));

复制代码

但QScopedPointer生命期结束后QScopedPointerPodDeleter (pod 是 “plain old data”的缩写) 会调用free来释放我们分配的空间。   
为了方便我们有一个专门针对数组的类，QScopedArrayPointer，在其生命期结束后会自动调用delete[]方法：

1. void foo()
2. {
3. QScopedArrayPointer<int> i(new int[10]);
4. i[2] = 42;
5. …
6. return; // our integer array is now deleted using delete[]
7. }

复制代码

注意如果你有一个引用计数的对象，可以使用QExplicitlySharedDataPointer来确保当其引用计数为0时正确删除。   
在Qt的S60分支中，QScopedPointe 和QExplicitlySharedDataPointer已经得到了广泛的使用。相信不久就可以加入Qt的总分支中。通过使用Qt的这些智能指针，我们可以让我们的程序更易读同时也不用过于担心，因为这些方法都是inline内联的。

经常这么用

class MyPrivateClass; // forward declare MyPrivateClass

class MyClass   
{   
private:   
    QScopedPointer<MyPrivateClass> privatePtr; // QScopedPointer to forward declared class

public:   
    MyClass(); // OK   
    inline ~MyClass() {} // VIOLATION - Destructor must not be inline

private:   
    Q\_DISABLE\_COPY(MyClass) // OK - copy constructor and assignment operators   
                             // are now disabled, so the compiler won't implicitely   
                             // generate them.   
};