

1+1=10 简简单单，我的小屋...

个人资料



dbzhang800

访问：1154532次  
积分：13428  
等级：  
排名：第356名

原创：181篇 转载：0篇  
译文：8篇 评论：834条

公告

本blog在IE浏览器下可能会格式错乱，请考虑非IE内核浏览器。

在多篇blog中，反斜杠被CSDN自动替换成了斜杠，请意识识别，谢谢。



本Blog中所有作品(文中额外注明版权的除外)均采用知识共享署名-非商业性使用-相同方式共享

CSDN Android客户端 正式发布 扒一扒最NB的开发项目 他们都已提交，就差你了！ 最流行的语言都在这，想学就学！

## Qt 智能指针学习

分类：C/C++ Qt 2011-05-08 12:17 16364人阅读 评论(13) 收藏 举报

qt delete object pointers reference destructor

目录(?) [+]

### 从内存泄露开始？

很简单的入门程序，应该比较熟悉吧 ^\_^

```
#include <QApplication>
#include <QLabel>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QLabel *label = new QLabel("Hello Dbzhang800!");

    label->show();

    return app.exec();
}
```

在 从 Qt 的 delete 说开来 一文中，我们提到这个程序存在内存泄露(表现就是析构函数不被调用)，而且当时给出了三种解决方法：

- 将label对象分配到stack而不是heap中
- 给label设置标记位Qt::WA\_DeleteOnClose
- 自己调用delete来删除通过new分配到heap中的 label 对象

[2.5](#)进行许可。

文章搜索

文章分类

- [C/C++](#) (53)
- [PyQt4/PySide](#) (13)
- [Python](#) (22)
- [Qt](#) (122)
- [QTBUG](#) (12)
- [tools](#) (33)
- [Python源码学习](#) (11)
- [C++0x](#) (5)
- [Qt5](#) (17)
- [Python Issue](#) (1)
- [git](#) (5)
- [cmake/qmake](#) (17)
- [QWidget漫谈](#) (13)
- [latex](#) (1)
- [javascript](#) (3)
- [Log4Qt](#) (3)
- [linux](#) (4)
- [C#/I.NET](#) (9)

Qt 博客链接

- [齐亮的博客](#)
- [白建平的博客](#)
- [白净的博客](#)
- [吴迪的博客](#)
- [我的blogspot](#)
- [我的百度博客](#)

文章存档

- [2013年07月](#) (1)
- [2012年11月](#) (1)
- [2012年05月](#) (2)
- [2012年04月](#) (1)
- [2012年03月](#) (4)

注：

为了能清楚观察构造和析构函数的调用，我们可以简单子类化了一下QLabel，然后用Label取代前面的QLabel

```
class Label :public QLabel
{
public:
    Label(const QString& text, QWidget *parent=NULL)
        :QLabel(text, parent){qDebug("from constructor");}
    ~Label(){qDebug("from destructor");}
};
```

本文中，我们从智能指针(smart pointer)角度继续考虑这个问题

## 智能指针

为了管理内存等资源，C++程序员通常采用RAII(Resource Acquisition Is Initialization)机制：在类的构造函数中申请资源，然后使用，最后在析构函数中释放资源。

如果没有智能指针，程序员必须保证new对象能在正确的时机delete，四处编写异常捕获代码以释放资源，而智能指针则可以在退出作用域时(不管是正常流程离开或是因异常离开)总调用delete来析构在堆上动态分配的对象。

我们看看Qt家族的智能指针：

智能指针		引入
QPointer	Qt Object 模型的特性(之一) 注意：析构时不会delete它管理的资源	
QSharedPointer	带引用计数	Qt4.5
QWeakPointer		Qt4.5
QScopedPointer		Qt4.6
QScopedArrayPointer	QScopedPointer的派生类	Qt4.6
QSharedDataPointer	用来实现Qt的隐式共享(Implicit Sharing)	Qt4.0
QExplicitlySharedDataPointer	显式共享	Qt4.4
std::auto_ptr		
std::shared_ptr	std::tr1::shared_ptr	C++0x

最新评论

如何让 Qt 的程序使用 Sleep  
muggle222: 我抄了楼主提供的代码: #ifdef Q\_OS\_WIN Sleep(uint(ms));#els...

从QProcess说开来(一)  
wh9959: 请问一下, 使用QProcess, 运行一个linux程序, 其程序的标准输出, 为什么一定要等这个程序停止...

Qt中的和字节流有关的几个Buffer  
边城菜鸟: 环形的buffer 我在QT里面 没弄出来。郁闷啊。

cmake 学习笔记(一)  
yc2011014265: 您好, 我在dos下执行camke .. -G"NMake Makefiles"的时候, 它报出一个错误...

C/C++ Strict Alias 小记  
FrankHB1989: strict aliasing rule在ANSI C89 3.3。C89章号小是因为ANSI和IS...

cmake 学习笔记(一)  
一动不动的葱头: 好得不得了!

QString之arg使用一则(QTBUG1  
天涯小小雨: 说好的最小的数字呢那为什么不是2% 而是20%

从QProcess说开来(二)  
baidu\_27059313: 您好 我写了一个简单的c++程序A.cpp, 能够成功输出“helloworld”, 现在我想在B.py...

Template+=fakelib 小记  
九万9w: @zhgn2:我也沮丧

Template+=fakelib 小记  
九万9w: Qt 中可不止这么一个潜规则

阅读排行

- cmake 学习笔记(一)  
(47466)
- Qt 线程基础(QThread、Q  
(29141)
- QTextCodec中的setCod  
(22508)
- 从 相对路径 说开来(从C-  
(19728)
- 从 Qt 的 delete 说开来  
(19163)
- 从QProcess说开来(一)  
(17241)
- qDebug 学习小结

std::weak_ptr	std::tr1::weak_ptr	C++0x
std::unique_ptr	boost::scoped_ptr	C++0x

注:

- MSVC2010 和 GCC g++ 4.3 支持 C++0x
- MSVC2008 sp1 及 GCC g++ 4.0 支持 tr1

有了这些东西, 我们就可以很容易改造我们前面的例子了(只需改变一行):

```
std::auto_ptr<QLabel> label(new QLabel("Hello Dbzhang800!"));
```

根据你所用的Qt的版本, 以及C++编译器的支持程度, 你可以选用:

- QScopedPointer
- std::unique\_ptr
- QSharedPointer
- std::shared\_ptr
  - std::tr1::shared\_ptr

## QPointer

如何翻译呢? 我不太清楚, 保留英文吧。

The QPointer class is a template class that provides **guarded pointers** to QObjects.

- 使用: 一个guarded指针, QPointer<T>, 行为和常规的指针 T \* 类似
- 特点: 当其指向的对象(T必须是QObject及其派生类)被销毁时, 它会被自动置NULL.
  - 注意: 它本身析构时不会自动销毁所guarded的对象
- 用途: 当你需要保存其他人所拥有的QObject对象的指针时, 这点非常有用

一个例子

```
QPointer<QLabel> label = new QLabel;  
  
label->setText("&Status:");  
  
...  
  
if (label)  
    label->show();
```

如果在...部分你将该对象delete掉了, label会自动置NULL, 而不会是一个悬挂(dangling)的野指针。

QPointer 属于Qt Object模型的核心机制之一, 请注意和其他智能指针的区别。

<a href="#">浅谈 qmake 之 pro、pri、</a>	(17180)
<a href="#">Qt 智能指针学习</a>	(16508)
<a href="#">QString 乱谈(3)-Qt5与中</a>	(16347)
	(16066)

评论排行	
<a href="#">用ISO C++实现自己的信</a>	(57)
<a href="#">从 Qt 的 delete 说开来</a>	(45)
<a href="#">cmake 学习笔记(一)</a>	(39)
<a href="#">1+1=2的 blog 文章索引</a>	(37)
<a href="#">从 相对路径 说开来(从C-</a>	(32)
<a href="#">QTextCodec中的setCod</a>	(27)
<a href="#">QString乱谈(2)</a>	(20)
<a href="#">Qt编码风格</a>	(19)
<a href="#">Qt国际化（源码含中文时</a>	(18)
<a href="#">QMainWindow上下文菜</a>	(18)

## std::auto\_ptr

这个没多少要说的。

- 不能让多个auto\_ptr 指向同一个对象！(auto\_ptr被销毁时会自动删除它指向的对象，这样对象会被删除多次)
  - 通过拷贝构造或赋值进行操作时，被拷贝的会自动变成NULL，复制所得的指针将获得资源的唯一所有权
- 智能指针不能指向数组(因为其实现中调用的是delete而非delete[])
- 智能指针不能作为容器类的元素。

在C++0x中，auto\_ptr已经不建议使用，以后应该会被其他3个智能指针所取代。

## QScopedPointer 与 std::unique\_ptr

它们概念上应该是是一样的。下面不再区分：

这是一个很类似auto\_ptr的智能指针，它包装了new操作符在堆上分配的动态对象，能够保证动态创建的对象在什么时候都可以被正确地删除。但它的所有权更加严格，不能转让，一旦获取了对象的管理权，你就无法再从它那里取回来。

无论是QScopedPointer 还是 std::unique\_ptr 都拥有一个很好的名字，它向代码的读者传递了明确的信息：这个智能指针只能在本作用域里使用，不希望被转让。因为它的拷贝构造和赋值操作都是私有的，这点我们可以对比QObject及其派生类的对象哈。

用法 (来自Qt的manual):

考虑没有智能指针的情况，

```
void myFunction(bool useSubClass)
{
    MyClass *p = useSubClass ? new MyClass() : new MySubClass;

    QIODevice *device = handsOverOwnership();

    if (m_value > 3) {
        delete p;
        delete device;
        return;
    }

    try {
        process(device);
```

```
    }

    catch (...) {

        delete p;

        delete device;

        throw;

    }

    delete p;

    delete device;

}
```

我们在异常处理语句中多次书写**delete**语句，稍有不慎就会导致资源泄露。采用智能指针后，我们就可以将这些异常处理语句简化了：

```
void myFunction(bool useSubClass)
{

    QScopedPointer<MyClass> p(useSubClass ? new MyClass() : new MySubClass);

    QScopedPointer<QIODevice> device(handsOverOwnership());

    if (m_value > 3)

        return;

    process(device);

}
```

另，我们一开始的例子，也是使用这两个指针的最佳场合了(出**main**函数作用域就将其指向的对象销毁)。

注意：因为拷贝构造和赋值操作私有的，它也具有**auto\_ptr**同样的“缺陷”——不能用作容器的元素。

## QSharedPointer 与 std::shared\_ptr

**QSharedPointer** 与 **std::shared\_ptr** 行为最接近原始指针，是最像指针的"智能指针"，应用范围比前面的提到的更广。

**QSharedPointer** 与 **QScopedPointer** 一样包装了**new**操作符在堆上分配的动态对象，但它实现的是引用计数型的智能指针，可以被自由地拷贝和赋值，在任意的地方共享它，当没有代码使用（引用计数为0）它时才删除被包装的动态分配的对象。**shared\_ptr**也可以安全地放到标准容器中，并弥补了**std::auto\_ptr** 和 **QScopedPointer** 因为转移语义而不能把指针作为容器元素的缺陷。

## QWeakPointer 与 std::weak\_ptr

强引用类型的QSharedPointer已经非常好用，为什么还要有弱引用的 QWeakPointer？

QWeakPointer 是为配合 QSharedPointer 而引入的一种智能指针，它更像是 QSharedPointer 的一个助手(因为它不具有普通指针的行为，没有重载operator\*和->)。它的最大作用在于协助 QSharedPointer 工作，像一个旁观者一样来观测资源的使用情况。

- weak\_ptr 主要是为了避免强引用形成环状。摘自msdn中一段话：

A cycle occurs when two or more resources controlled by shared\_ptr objects hold mutually referencing shared\_ptr objects. For example, a circular linked list with three elements has a head node N0; that node holds a shared\_ptr object that owns the next node, N1; that node holds a shared\_ptr object that owns the next node, N2; that node, in turn, holds a shared\_ptr object that owns the head node, N0, closing the cycle. In this situation, none of the reference counts will ever become zero, and the nodes in the cycle will not be freed. To eliminate the cycle, the last node N2 should hold a weak\_ptr object pointing to N0 instead of a shared\_ptr object. Since the weak\_ptr object does not own N0 it doesn't affect N0's reference count, and when the program's last reference to the head node is destroyed the nodes in the list will also be destroyed.
- 在Qt中，对于QObject及其派生类对象，QWeakPointer有特殊处理。它可以作为QPointer的替代品
  - 这种情况下，不需要QSharedPointer的存在
  - 效率比 QPointer 高

## QSharedDataPointer

这是为配合 QSharedData 实现隐式共享(写时复制 copy-on-write))而提供的便利工具。

Qt中众多的类都使用了隐式共享技术，比如QPixmap、QByteArray、QString、...。而我们为自己的类实现隐式共享也很简单，比如要实现一个 Employee类：

- 定义一个只含有一个数据成员(QSharedDataPointer<EmployeeData>) 的 Employee 类
- 我们需要的所有数据成员放置于 派生自QSharedData的 EmployeeData类中。

具体实现看 QSharedDataPointer 的Manual，此处略

## QExplicitlySharedDataPointer

这是为配合 QSharedData 实现显式共享而提供的便利工具。

QExplicitlySharedDataPointer 和 QSharedDataPointer 非常类似，但是它禁用了写时复制功能。这使得我们创建的对象更像一个指针。

一个例子，接前面的Employee：

```
#include "employee.h"

int main()
{
    Employee e1(1001, "Albrecht Durer");
    Employee e2 = e1;
    e1.setName("Hans Holbein");
}
```

写时复制技术导致：**e1**和**e2**有相同的工号，但有不同名字。与我们期待的不同，显式共享可以解决这个问题，这也使得**Employee**本身更像一个指针。

## 补遗

先前竟未注意到官方的这两篇文章（这是失败）：

- <http://labs.qt.nokia.com/2009/08/25/count-with-me-how-many-smart-pointer-classes-does-qt-have/>
- <http://labs.qt.nokia.com/2009/08/21/introducing-qscopedpointer/>

便看看google编码规范中对3个智能指针的建议：

scoped\_ptr

Straightforward and risk-free. Use wherever appropriate.

auto\_ptr

Confusing and bug-prone ownership-transfer semantics. Do not use.

shared\_ptr

Safe with const referents (i.e. shared\_ptr<const T> ). Reference-counted pointers with non-const referents can occasionally be the best design, but try to rewrite with single owners where possible.

## 参考

- Qt manual
- [Smart Pointers FAQ](#)
- [The New C++:Smart\(er\) Pointers](#)
- [http://en.wikipedia.org/wiki/Smart\\_pointer](http://en.wikipedia.org/wiki/Smart_pointer)
- [C++:智能指针-TR1的shared\\_ptr和weak\\_ptr使用介绍](#)
- [【C++Boost】智能指针的标准之争：Boost vs. Loki](#)
- <http://www2.research.att.com/~bs/C++0xFAQ.html>
- <http://msdn.microsoft.com/zh-cn/library/bb982126.aspx>



- <http://www.codesynthesis.com/~boris/blog/2010/05/24/smart-pointers-in-boost-tr1-cxx-x0/>
- <http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml>

上一篇 使用Shiboken为C++和Qt库创建Python绑定

下一篇 Qt 状态机框架学习

主题推荐      智能指针      内存泄露      异常处理      initialization      程序员

猜你在找

- 基于Qt的多窗口设计B-窗体切换的实现

如何让 Qt 的程序使用 Sleep

Qt学习停靠窗体QDockWidget类

用qwt绘制AD波形转载

wchar\_t内置还是别名小问题一则
- 【精品课程】JavaScript for Qt Quick(QML)

【精品课程】Qt基础与Qt on Android入门

【精品课程】火星人敏捷开发1001问（第二季）


【精品课程】零基础学Java系列从入门到精通

【精品课程】微信公众平台开发入门

准备好了么？跳吧！

更多职位尽在 CSDN JOB

高级嵌入式软件工程师（智能家居、物联	我要跳槽	Android智能应用开发工程师	我要跳槽
翔傲信息科技（上海）有限公司	10-20K/月	衡阳市凯讯科技有限公司	4-8K/月
项目经理（可穿戴智能产品方向）	我要跳槽	算法工程师-机器学习方向	我要跳槽
浩沙实业(福建)有限公司	8-12K/月	北京运科网络科技有限公司	15-30K/月

美的(Midea) MT10NE-AA 电烤箱 家用迷你 多功能烘焙...

CN¥99.00 网购上京东，多、快、好、省！

京东[www.JD.com](http://www.JD.com)

查看评论

13楼 [lys211](#) 2015-03-14 22:05发表



我还需要消化啊，谢谢

12楼 [hanvash](#) 2014-06-06 17:19发表



[e01]

11楼 [木容峰](#) 2013-08-08 13:51发表





不错

10楼 Rainr 2013-08-01 18:58发表



消化消化。。

9楼 lpp1989 2012-04-25 13:54发表



我还需再消化消化。谢谢！

8楼 oneofzero 2011-08-26 17:07发表



很好！

7楼 chenzhp 2011-05-30 15:01发表



[e01][e01]

6楼 kenanhcf 2011-05-13 17:44发表



[e01][e01][e01][e01][e01][e01][e01][e01][e01][e02]

5楼 trcj1 2011-05-13 13:05发表



[e10]

4楼 luoye 2011-05-13 09:28发表



[e01][e01]

3楼 hucheng2009 2011-05-12 16:32发表



[e02]

2楼 hucheng2009 2011-05-12 16:31发表



[e03][e03]

1楼 PigiRoNs 2011-05-10 23:50发表



[e01][e01][e01]

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

\* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题

VPN

Hadoop

Spark

AWS

ERP

移动游戏

IE10

Eclipse

Java

CRM

Android

JavaScript

iOS

数据库

Swift

Ubuntu

智能硬件

NFC

Docker

WAP

OpenStack

jQuery

BI   HTML5   Spring   Apache   .NET   API   HTML   SDK   IIS   Fedora   XML   LBS   Unity  
Splashtop   UML   components   Windows Mobile   Rails   QEMU   KDE   Cassandra   CloudStack  
FTC   coremail   OPhone   CouchBase   云计算   iOS6   Rackspace   Web App   SpringSide   Maemo  
Compuware   大数据   aptech   Perl   Tornado   Ruby   Hibernate   ThinkPHP   HBase   Pure   Solr  
Angular   Cloud Foundry   Redis   Scala   Django   Bootstrap

[公司简介](#) | [招贤纳士](#) | [广告服务](#) | [银行汇款帐号](#) | [联系方式](#) | [版权声明](#) | [法律顾问](#) | [问题报告](#) | [合作伙伴](#) | [论坛反馈](#)

[网站客服](#)   [杂志客服](#)   [微博客服](#)   [webmaster@csdn.net](mailto:webmaster@csdn.net)   400-600-2320 | 北京创新乐知信息技术有限公司 版权所有 | 江苏乐知网络技术有限公司 提供商务支持

京 ICP 证 070598 号 | Copyright © 1999-2014, CSDN.NET, All Rights Reserved 