文章编号:1671-4598(2019)07-0036-04 DOI:10.16526/j.cnki.11—4762/tp.2019.07.008 中图分类号:V249.32 文献标识码:A

基于 Qt 的软件内存泄漏静态检测技术研究

匡海燕，张玉中，刘仁千，李国杰，谷 威
（许继电气股份有限公司，河南 许昌 461000）

摘要：Qt 继承了 C++语言动态分配内存机制，保证了开发人员能根据实际需要灵活地使用内存，同时 Qt 也不可避免的要面对“内存泄漏”这个严重威胁软件安全的问题，虽然 Qt 采取了半自动化内存管理机制等措施，但不能从根本上解决问题；对此，提出了一种基于 Qt 的软件内存泄漏静态检测方法，该方法针对 Qt 的半自动化内存管理机制，通过静态分析被测对象中分配内存的代码识别出是否属于 Qt 自动管理的范围，从而准确地检测出内存泄漏和内存重复释放问题；并基于该检测方法设计了一种 Qt 内存泄漏自动检测工具，该工具能很大程度上提高测试效率。

关键词：Qt 内存管理机制；内存泄漏；内存重复释放；静态检测技术

Research on Static Detection Technology of Software Based Qt Memory Leak

Kuang Haiyan, Zhang Yuzhong, Liu Renqian, Li Guojie, Gu Wei
(XJ Electric Co., Ltd., Xuchang 461000, China)

Abstract: Qt inherits C++ language dynamic allocating memory management mechanism, it ensures that the developers can flexible use memory according to the actual needs. Meanwhile Qt has to face the “memory leak” problem of serious threating to software security. Although Qt adopted measures such as semi-automatic memory management mechanism, it has not fundamentally solved the problem. Therefore, this paper proposes a static detection method of software based Qt memory leak. For this semi-automatic memory management mechanism of QT, this method can identify whether belongs to the scope of Qt automatic management by statically analyzing these codes where the tested object allocated memory. Thereby it can accurately detect memory leak and memory repeated release. And based on this detection method, an automatic Qt memory leak detection tool is designed. This tool can greatly improve the test efficiency.

Keywords: Qt memory management mechanism; memory leaks; memory repeated release; static detection technology

0 引言

C++中堆内存的泄漏就是业界通称的内存泄漏。堆内存的特点是允许编程人员自由分配内存大小，但编译器不会自动释放，需要编程人员自己释放，也就是“谁申请，谁管理”，堆内存的这个特点方便编程人员灵活申请内存的同时，也给内存管理带来了隐患。编程人员需要申请内存时，只需要使用 malloc，new 等内存申请函数，如果函数执行成功，就能顺利从堆中分配到所需要的内存。但申请的内存在使用完后，一定要有对应的释放内存操作，释放内存的函数主要有 free、delete 等。如果没有释放，就会发生内存泄漏，如果多次释放会导致更严重的问题。内存泄漏的发生往往会影响程序性能；更有甚者影响到整个软件运行环境，如果内存泄漏问题严重到耗尽系统的内存资源，整个软件系统都会瘫痪，那造成的后果是无法估量的。如果内存释放多了，会导致软件莫名奇妙的问题，因为多释放的那块内存存储的信息是随机的。总之，内存问题至关重要，这关系到系统能否安全可靠运行的问题。

收稿日期:2018-12-17； **修回日期**:2019-01-22。
基金项目:国家电网许继集团重点项目(5292C0170039)。
作者简介:匡海燕(1978-),女,湖南衡阳人,主要从事软件测试技术方向的研究。

Qt 继承了 C++语言动态分配内存机制，保证了开发人员能根据实际需要灵活地使用内存，同时 Qt 也不可避免的要面对“内存泄漏”这个严重威胁软件安全的问题，虽然 Qt 采取了半自动化内存管理机制等措施，但不能从根本上解决问题。针对内存泄漏问题，市场上出现了很多针对 C++内存泄漏检测的工具，由于 Qt 的特殊性，针对 C++的内存检测工具，不适应 Qt 程序的测试。所以，针对 Qt 的内存泄漏检测方法及工具问题已成为当前测试领域的一个急需解决的问题。

1 Qt 内存管理机制

Qt 是一个跨平台的基于 C++ 编程语言的图形用户界面应用程序框架，该框架提供给应用程序开发者建立艺术级的图形用户界面所需的按钮、滚动条、菜单及其他对象等功能[1]。我们在读 Qt 程序时不难发现，Qt 程序中申请内存的地方很多，但很少出现 delete 等有程序开发人员主动释放的函数。其实，详细读代码会进一步发现 Qt 程序有自己的特点，凡是程序开发人员没有手动释放的内存，指向它的指针都是 QObject 类或该类的继承类。通过深入研究 Qt 内存管理发现，Qt 本身有一套半自动化管理机制。该机制有一个显著特点是，只要程序开发人员申请的内存交给 QObject 类或继承类托管，就不需要人工释放，QObject 类

中有自动释放堆内存的机制。但同时要求，一旦申请的内存交给托管人，就不能再人为释放，释放就会错误。

Qt 的半自动化管理机制还有另一种体现形式父子继承关系，比如，一个类 A 的子类 B 申请了一块堆内存，如果类 A 释放了内存，子类 B 就不用再释放，就是父类可以统一托管、释放子类的内存。除此之外，Qt 的内存管理还有其他几个特点。(1) Qt 针对 QWidget 类及其子类有一个特殊的内存自动释放标志位，在申请 QWidget 类及子类的内存时只要设置了 Qt::WA _ DeleteOnClose 标志位，就不用再手动释放内存了。(2) 针对 QAbstractAnimation 类及其子类有一个特殊的内存自动释放标志位，在申请 QAbstractAnimation 类及子类的内存时只要设置了 QAbstractAnimation::DeleteWhenStopped 标志位，就不用再手动释放内存了。(3) 针对 QRunnable 类及其子类有一个特殊的内存自动释放标志位，在申请 QRunnable 类及子类的内存时只要设置了 QRunnable::setAutoDelete () 和 MediaSource::setAutoDelete () 标志位，就不用再手动释放内存了。

文献 [2-4] 在 2017 年提出的一种基于 Qt 的系统内存泄漏检测方法只解决了 QObject 的类及其继承的类相关的内存泄漏检测问题，还有相当一大部分 Qt 内存泄漏问题没有解决。文中详细给出了 Qt 内存泄漏的静态检测方法，并基于该方法设计开发了静态检测工具。

2　Qt 内存泄漏检测方法

在 Qt 程序中，如果发现一个 malloc，realloc，new 等函数，首先检查该函数中是否包含 WA _ DeleteOnClose、DeleteWhenStopped、setAutoDelete ()、parent 等关键字。如果有这些关键字，则不需要再进行分析下去，可以确定该处的内存不需要人工释放；如果有这些关键字，又发现对应的 free 或 delete，则认为是重复释放内存问题，属于严重问题。如果没有发现这些关键字，接着检查使用该函数申请内存的对象是否有父控件[5-8]，如果有父控件则也认为该处申请的内存不需要人工释放，如果发现人工释放的 free 或 delete 函数，认为是重复释放内存问题。如果既没有 parent、WA _ DeleteOnClose、DeleteWhenStopped、setAutoDelete () 等关键字，也没有发现父控件。则按 C++ 的内存泄漏检测方法检查，申请多少内存，用后一定要释放多少内存，一定要确保申请与释放的内存，大小相等，位置相同，否则认为内存处理异常。该内存泄漏检测方法的具体流程如图 1 所示。

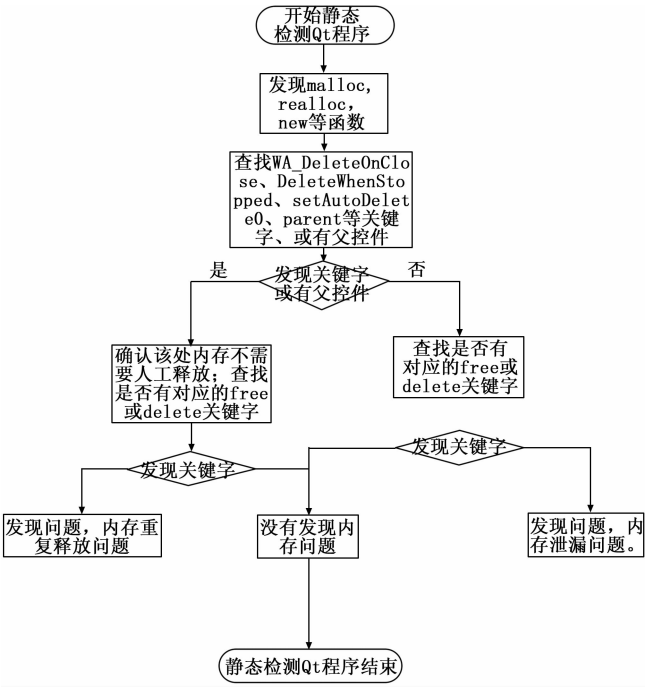

图 1　Qt 程序内存泄漏检测方法实现图

3　基于 Qt 内存泄漏检测方法的工具设计与实现

3.1　静态检测工具的设计

基于文中所提出的 Qt 内存泄漏的检测方法设计开发了测试工具。

3.1.1　静态测试工具的功能设计

静态测试工具的功能结构如图 2 所示，工具主要有静态分析和结果展示两个大模块，其中，静态分析模块包含文件分析模块、工程分析模块两个子模块；结果展示模块包含界面展示模块、问题报告生成模块等模块。

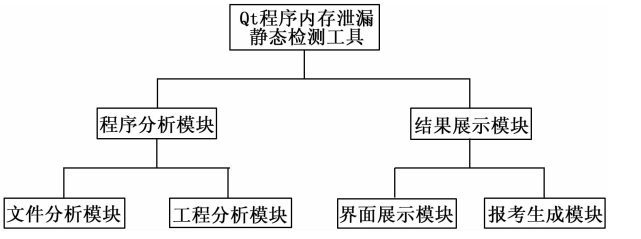

图 2　静态检测工具功能结构图

该静态检测工具的程序分析模块主要功能是实现支持单个 Qt 程序文件（单个 .cpp 或 .c 文件）加载和支持整个工程加载，这里主要支持 QtCreator 工程加载；如果是 QtCreator 工程加载，其分析过程相对复杂些，首先，对工程的文件类型进行分类，找出需要分析的 .cpp 文件和 .h 文件，然后对 .cpp 文件进行重点分析，通过逐行扫描 .cpp 文件，找出文件中的变量信息、函数信息、程序语句结构信息和内存信息。其中，变量信息主要是指变量的类型，变量的赋值情况等；函数信息主要是指函数的调用关系、函数的类型，参数等；程序语句结构信息主要是指程序的数据流程等关键信息；内存信息就是本文所重点关注的信息，主要是指内存泄漏或内存重复释放等问题。在扫描 .cpp 文件的过程中，如果有复杂的结构体、类等需要头文件的，直接查找相应的头文件进行分析获取相关信息。如果是 .cpp 文件加载，其分析过程相对简单，只需要重复工程分析中对 .cpp 文件的分析过程即可，这里不在赘述。需要说

明的是再加载 .cpp 文件的时候，如果有对应的头文件，也一并加载上，以保证分析程序的顺利进行。

结果展示模块的主要功能是实现对分析结果的界展示，从展示形式上又划分为界面形式的展示和报告形式的展示。其中，界面形式的展示主要包括：1）被测源代码及定位问题的展示，主要方便测试人员现场分析并定位问题；2）变量信息的集中展示，包括变量是需要赋值的或给其他变量的赋值的信息等，主要方便测试人员对整体被测程序变量的分析特别是整个 QtCreator 工程加载时，特别有用；3）函数调用关系的展示，可以方便测试人员对整个测试源代码的整体函数关系有个了解，便于分析和理解程序；4）程序语句结构展示，主要是程序数据流程图的展示，方便测试人员进一步动态测试用。5）内存问题展示，这个是界面展示功能的核心所在，主要是展示工具静态分析中发现的疑似内存泄漏或内存重复释放的问题，以供测试人员分析。报告生成模块主要是对工具发现的疑似问题生成一个 Word 文档的报告，用于报告形式展示静态分析的结果。

3.1.2 静态测试工具的工作流程设计

静态测试工具的工作流程如图 3 所示，加载单个文件的测试流程与加载工程的测试流程的处理有所区别。加载对象为一个工程时，首先分析工程，把工程中的所有文件分类保存在不同的链表中，然后在进一步处理链表中的节点信息。当加载的是单个. cpp（或. c）文件时，需要指定. cpp 文件所对应的头文件路径，然后再对文件进行静态分，该流程的具体实现将在工具的实现中做详细描述。

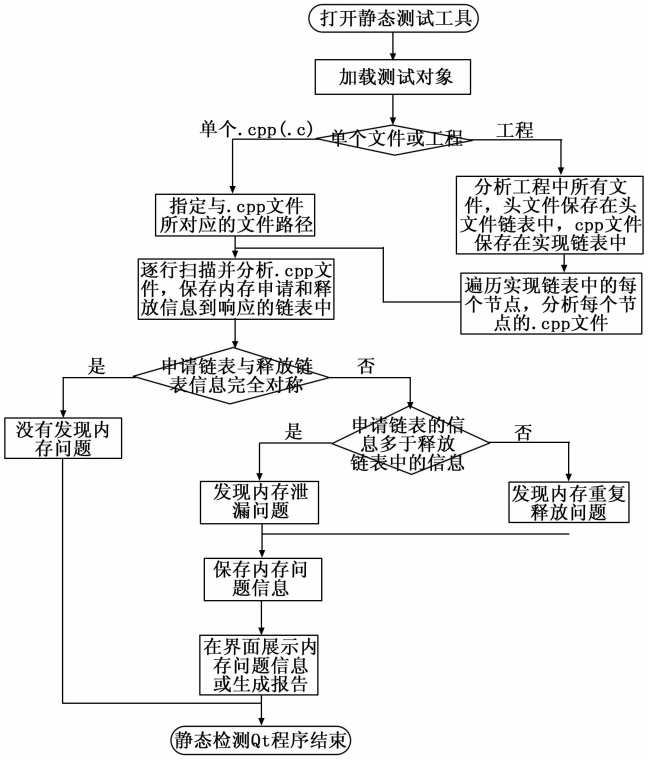

图 3 静态检测工具的工作流程图

3.2 静态检测工具的实现

静态检测工具是在 Linux 系统下采用 Qt/C++开发实现的，开发环境使用 Qt Creator。主界面如图 4 所示，主菜单中有文件、静态分析测试和质量评价模型三项。其中，文件菜单中包含设置头文件目录、加载 C 文件和加载工程文件。因为静态分析工具加载被测对象时支持单个 C 文件加载，所以需要对应的头文件需要设置。静态分析测试项中不仅包含内存泄漏情况分析，还包括静态全局分析、函数关系分析等源代码其他以供质量度量时需要的度量指标信息和生成测试报告。质量度量模型项主要包括质量度量元信息饼图、质量度量标准饼图和质量度量因素饼图的展示子项。

图 4 静态测试工具主界面

总之，该静态工具的实现中，不仅包含了内存泄漏静态分析的功能，还有还包含了质量度量的功能。在实际应用过程中，执行内存静态分析时，后期测试和质量度量的其他信息都一起分析统计了，这样进一步提高了整体的测试效率。

以下对静态分析工具的核心功能给出具体的实现。

3.2.1 加载分析单个文件的功能实现

文中所设计的静态分析工具支持对单个文件（.cpp 或 .c 文件）的加载并分析，这里重点介绍工具该项功能的实现。

对加载文件进行逐行扫描，取得文件中的 new、malloc 等关键字放到一个内存申请链表中，并进一步判断 new、malloc 处理的对象是否有 delete、free 关键字的对应处理，如果有，则放到内存释放链表中；如果没有对应的处理，则继续判断是否是属于 Qt 的内存托管，具体托管的种类，依据前面提到的 Qt 内存半自动化管理分类，并把对应的 Qt 内存托管类型存储到内存释放链表中。文件扫描分析结束后，通过对比内存申请链表与内存释放链表的信息，得出程序内存释放存在内存泄漏的情况。

另外，在对文件进行逐行分析的过程中，如果遇到数据类型为结构体或更复杂的类时，需要进一步分析与 .cpp（或 .c）文件对应的头文件时，分析工具可以通过用户指定路径的方法自动加载并分析所对应的头文件。

3.2.2 加载分析工程的功能实现

工具加载并分析工程的功能主要是扫描并分析工程里的所有 .c，.cpp 和 .h 文件。首先把所有 .h 文件放到一个头文件链表中，把所有 .c，.cpp 文件放到实现链表中。然

后，遍历实现链表中的每个节点，并对每个节点中的文件进行逐行扫描分析，分析的内容与 2.2.1 中对单个文件的分析过程及内容相同，这里不再赘述。需要说明的是这里每个节点分析出的内存申请链表和内存释放链表，需要与文件链表中的节点关联起来，关联关系如图 5 所示，实现文件链表中，每个节点对应两个内存申请和释放信息链表。通过对比内存申请链表与内存释放链表的信息，得出程序内存释放存在内存泄漏的情况。

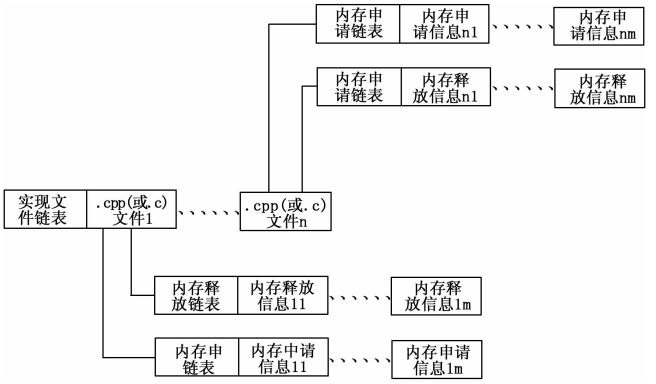

图 5　实现文件的链表信息结构图

另外，在对每个. cpp（或. c）文件进行逐行分析的过程中，如果遇到数据类型为结构体或更复杂的类时，需要进一步分析与. cpp（或. c）文件对应的头文件时，分析工具通过遍历头文件链表并分析所对应的头文件。

4　工具实际应用及效果

文中所设计开发的 Qt 程序内存泄漏检测工具在某公司重点项目的白盒测试中得到了广泛应用。工具支持 Linux 和 window 操作系统，本次应用主要运行在 Linux 系统上，静态分析工具安装在一台 PC 机上，PC 机具体软硬件配置信息如下表 1 所示。

表 1　PC 机软硬件配置信息

测试机	浪潮 NF5280 M2
物理内存大小	4G
CPU	Intel Xeon(R)CPU E5606 @2.13GHZ * 8
内核版本	3.10.0－514.2.2el7.x86_64
操作系统	Red Hat Enterprise Linux Server 7.3
网卡带宽	1000Mbps

通过对几十个模块的多轮次的迭代测试，发现内存泄漏问题近百个，并且都能准确定位，经验证，问题误报和漏报率都少于 1%。工具发现的 Qt 内存泄漏问题主要总结为两类：第一类是未管理的 Qt 内存发生泄漏；第二类是 Qt 半自动管理的内存被再次释放。比如申请内存时，构造函数不带参数，即未指定父亲，会引发内存泄漏；申请内存时指定了父亲却被再次释放导致错误。除此以外，还有一些特殊情况，比如指定父亲为 this 指针、申请的内存交给全局变量管理、用增加孩子 addChild 的方式交给父亲管理内存、申请的内存永久性加入主窗口等，这些情况都相当于间接指定了父亲，也可以不用考虑内存泄漏。工具准确定位的 Qt 内存泄漏问题，为研发人员的编码工作提供了有效的参考，得到测试和研发人员的高度认可。

在测试效率方面的效果更显著，能帮助测试人员快速排除非内存泄漏问题，比如很多关于 Qt 内存半自动化管理范围内的程序，测试人员人工分析排除非内存泄漏问题时，需要耗费大量的时间。现在工具可以直接根据文中所提出的 Qt 内存泄漏测试方法进行自动排除，很大程度上提高了测试效率，如图 6 所示，人工分析与工具扫描一个模块源代码的平均时间对比图。

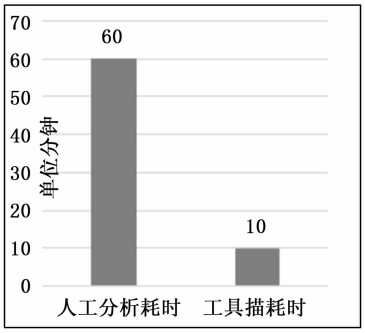

图 6　人工分析与工具扫描耗时对比图

5　结论

通过对 Qt 内存半自动化管理的分析，提出了基于 Qt 程序内存泄漏检测方法，并设计开发了与该方法相对应的内存泄漏静态分析测试工具，经过集团公司某重点项目的具体应用发现，该工具切实有效，在提高测试效率和内存泄漏问题代码精准定位方面效果明显。

参考文献：

[1] Blanchette J，Summerfield M. C＋＋ GUI programming with Qt 4 [M]. 2nd ed. [s. l.]: [s. n.]，2015.

[2] 张　玲，李　艳，胡　术，等. 一种基于 Qt 的系统内存泄漏检测方法 [J]. 计算机技术与发展，2017，27（12）：120－126.

[3] 汪小林，王振林，孙逸峰，等. 利用虚拟化平台进行内存泄露探测 [J]. 计算机学报，2010，33：463－472.

[4] 李晓南，范明钰，王光卫. 基于静态检测工具的源代码安全缺陷检测技术 [J]. 计算机应用研，2011，28（8）：2997－2998.

[5] 柳　青，杨英豪，孙永超. C＋＋内存检测的研究 [J]. 电子工业专用设备，2014，43（12）：35－38.

[6] 张　鹏，杨秋辉，李海怒. 嵌入式软件内存泄露检测方法研究 [J]. 计算机工程与应用，2013，27（11）：38－42.

[7] Bush W R，Princus J D，Sielaff D J. A static analyzer for finding dynamic programming errors [J]. Software Practice and Experience，2000：775－800.

[8] 吕维梅，刘　坚. C/C＋＋程序安全漏洞的分类与分析 [J]. 计算机工程与应用，2003，39（6）：37－40.