Qt性能优化方案介绍

林斌
13918814219
richard.lin@qt.io

2018.09.11

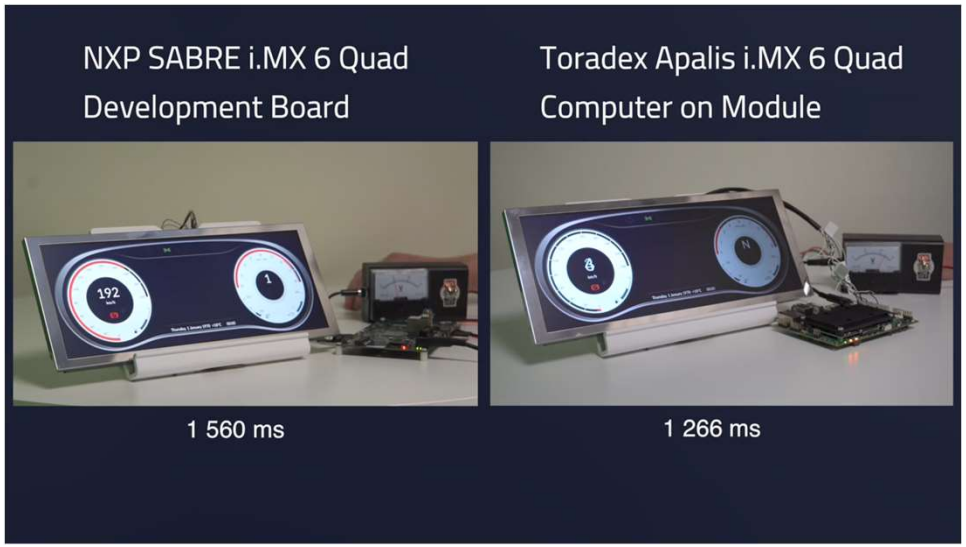
FastBoot
QML程序设计优化
多线程编程

15:45 – 16:30 0.75小时


Qt
Fastboot

快速启动 汽车数字仪表盘 演示

通过深度优化满足您的需求



硬件软件配置

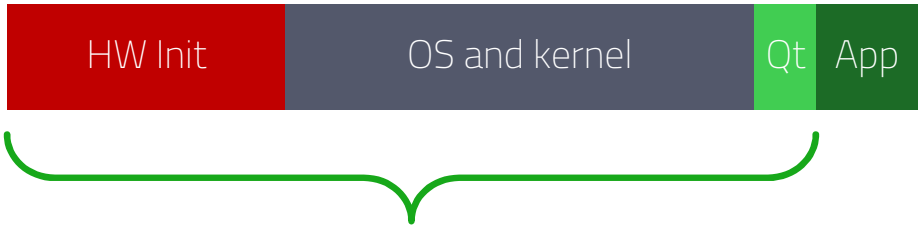
+ i.MX6 Quad 1G
+ 1G Memory
+ Qt 5.6/embedded linux
+ Qt for device creation
+ 12.3″ HSXGA (1280*480)

完成优化项目

+ customization of boot loader
+ reduced Qt libraries
+ Static linking
+ Qt Quick Compiler
+ dynamic loading of QML items

优化结果

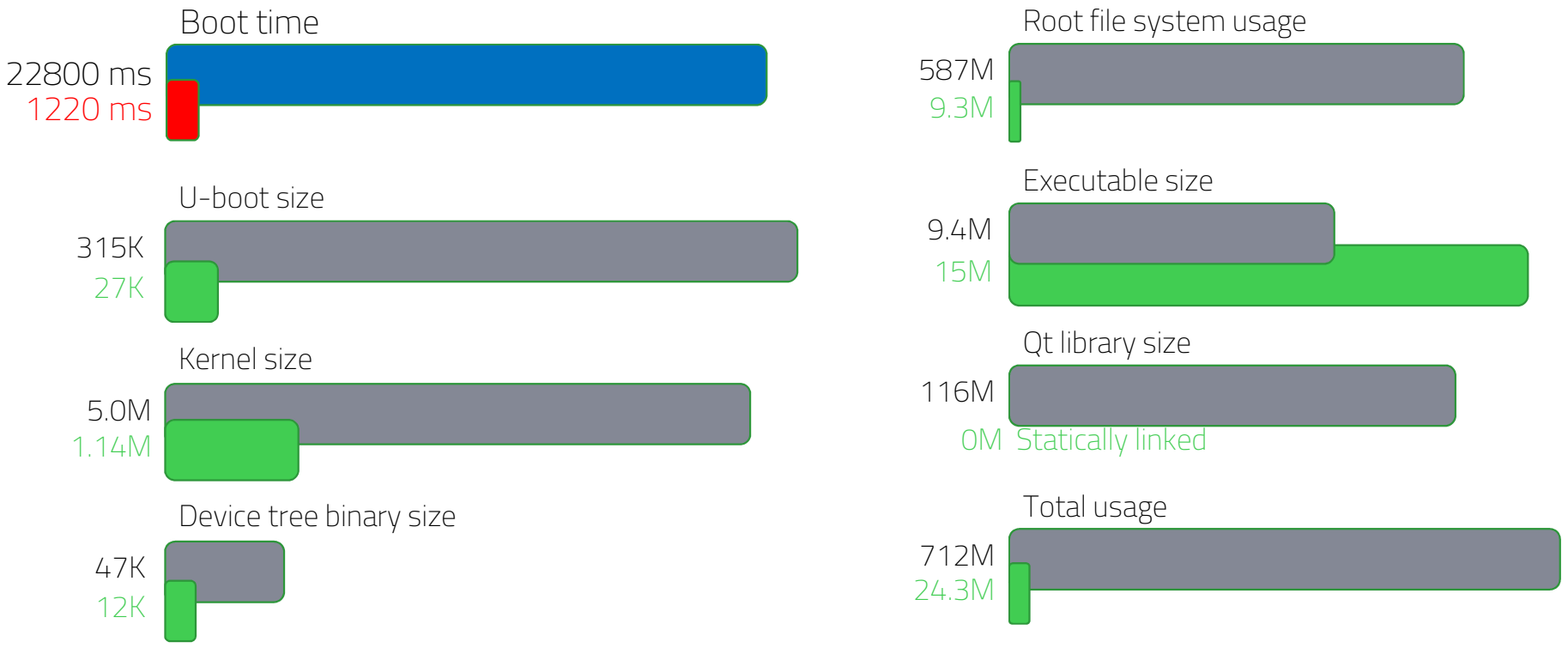
正常启动时间 23 sec
优化后的启动时间 1.56 sec



Boot time from power on to visible UI on screen

快速启动 汽车数字仪表盘 总结

使用了Qt Lite、静态链接、QML优化以及操作系统优化



Fastboot 优化何处着手?

- 库和编译优化
  - Qt库的裁剪
  - 静态链接
  - QML Quick Compiler
- 应用优化
  - 良好的风格与习惯
  - 多线程编程
  - QML编程技巧

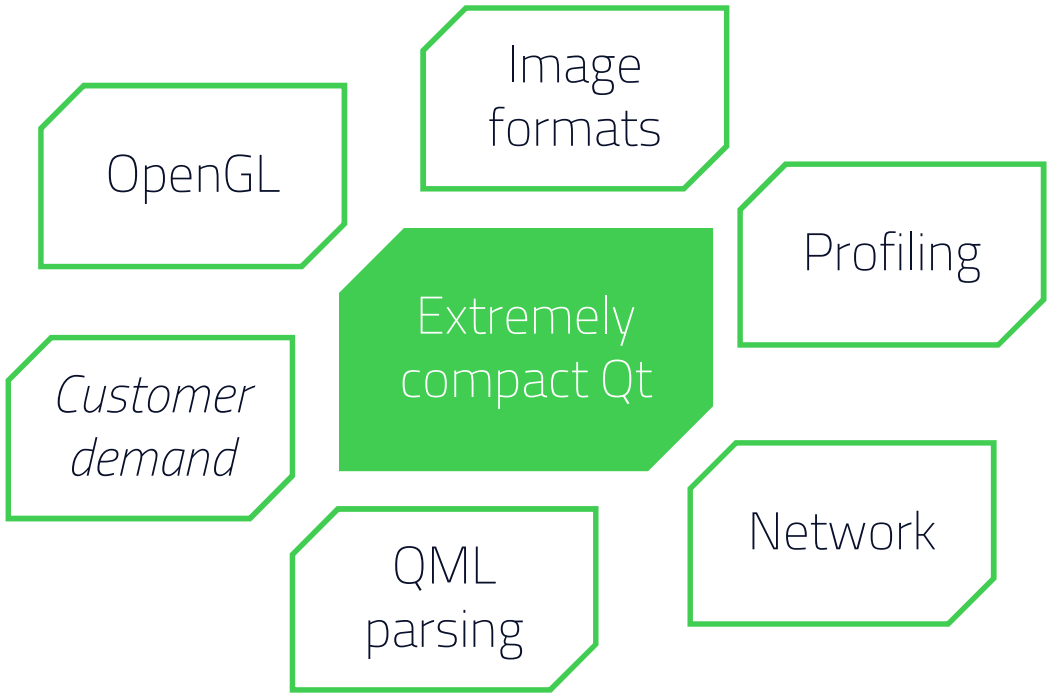
Qt库的裁剪

- 遇到的问题
  - 模块、函数之间的耦合性
  - 裁减掉Qt部分模块对一般程序员很困
  - 缺少在资源有限的环境下创建系统的标准手册
- Qt Lite project
  - 方便Qt的裁剪

Introduced in Qt 5.8

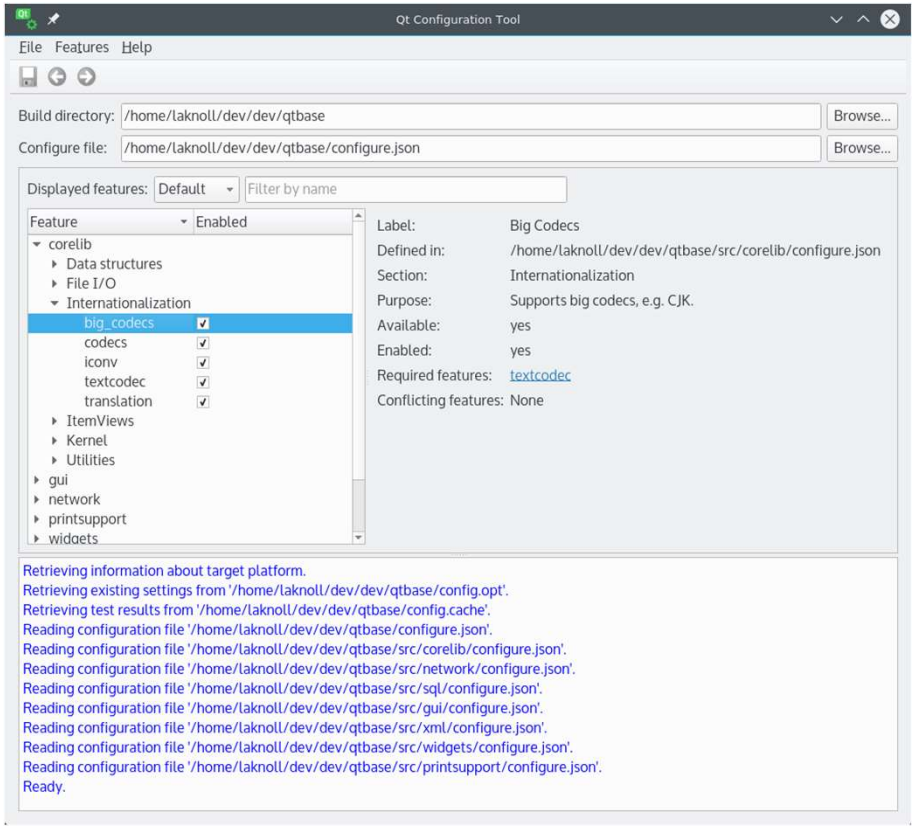
Qt Lite Project

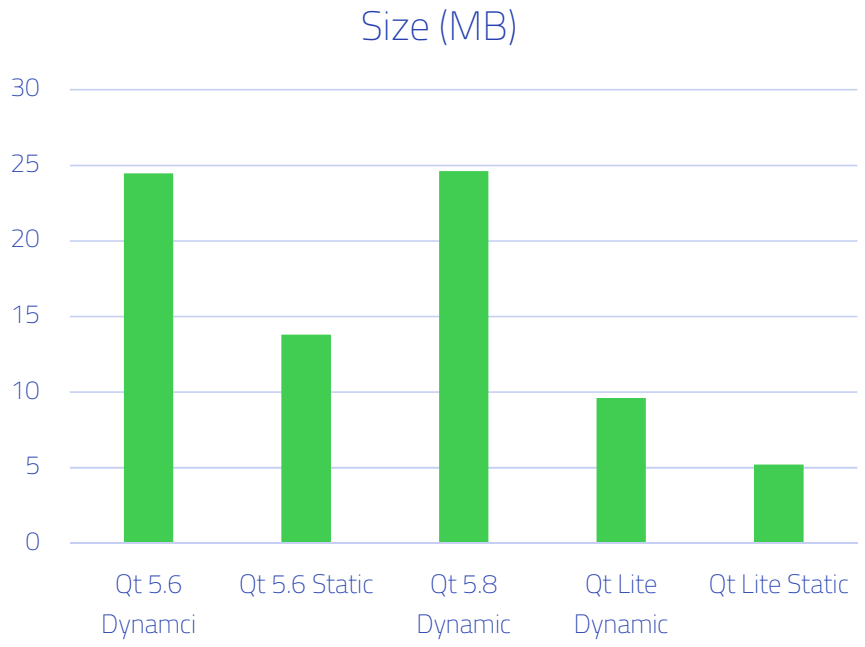
- Qt Lite project的目标
  - 更小的库尺寸(< 10 MB)
  - 最小的系统和应用加载事件
  - 高灵活性和可配置性
  - 图形化配置界面



Introduced in Qt 5.8

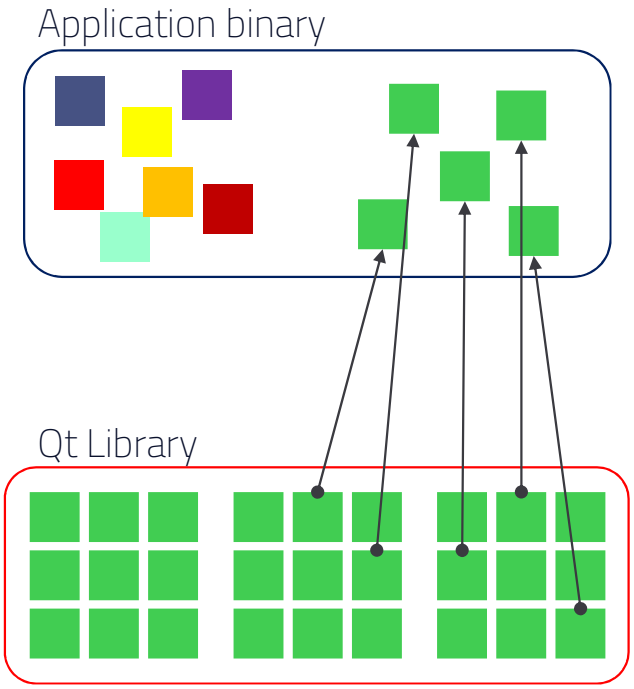
The Qt Lite – Configuration tool





静态链接

- 缩短应用加载时间
  - 静态链接减少程序加载时间
  - 确保恒定的加载时间
- 灵活定制用户界面和类库
  - 最佳优化内存占用
  - 精简Qt类库
- 高性能
  - 仅集成需要的库
  - 单进程架构获得最高性能
- 安全性
  - 避免动态链接库被恶意替换
- 兼容性
  - 没有预装库的兼容问题



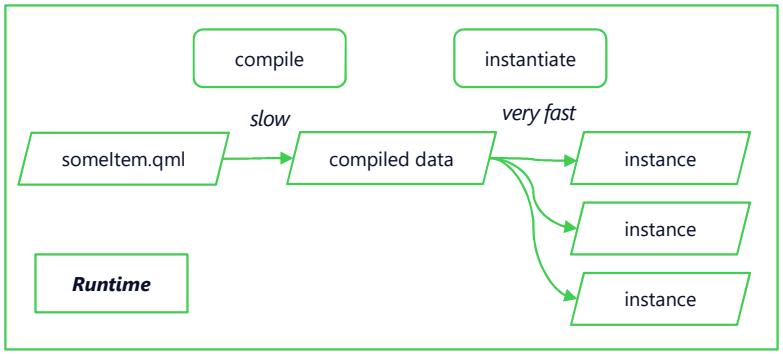
Introduced in Qt 5.3

Qt Quick Compiler

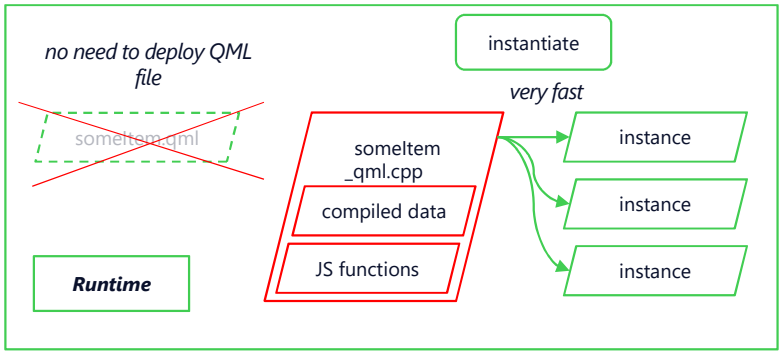
将QML源码编译至二进制码
只需增加一行代码就能确保更快运行

- 好处
  - 加快20-30%的QML加载时间
  - 缓存经过JIT编译的代码，更快地执行
  - 在不支持JIT的平台上也能提升性能
  - 代码安全性
- 方便易用
  - 只需要在PRO文件中添加一行代码：
    CONFIG += qtquickcompiler

开源版QML



商业版Qt Quick Complier



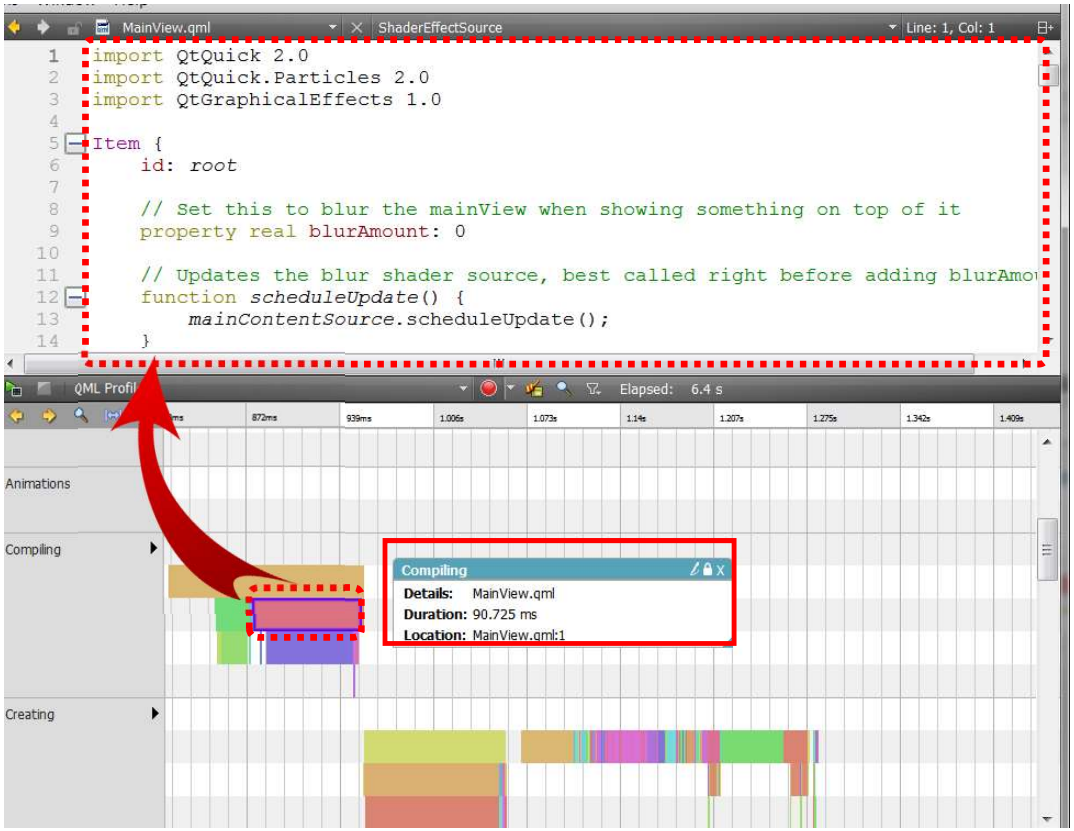
QML程序设计优化

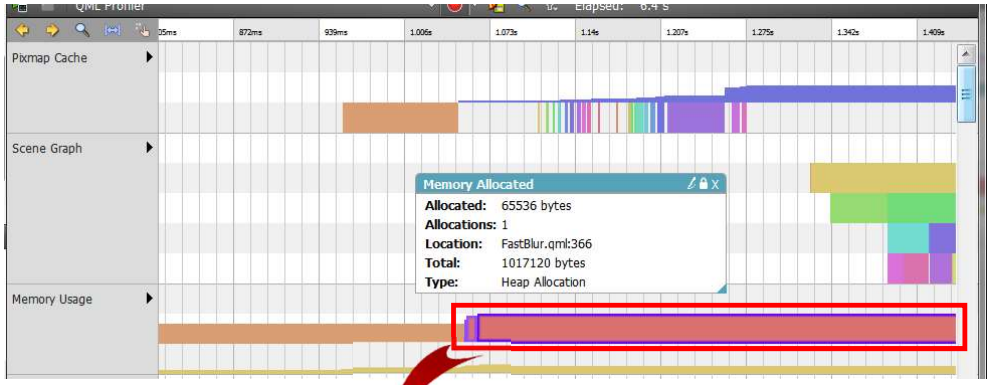
Qt Quick性能优化
目标60Hz

- 使用事件驱动
  - 避免定时轮询
  - 使用信号槽形式
- 使用多线程
  - C++
  - QML WorkerScript元件
- 使用Qt Quick Compiler
  - 只需要在PRO文件中添加一行代码:
  - CONFIG += qtquickcompiler
- 避免使用CPU渲染的元件
  - Canvas、Qt Charts
- 使用异步加载
  - 图片异步加载
  - 使用C++处理大数据加载

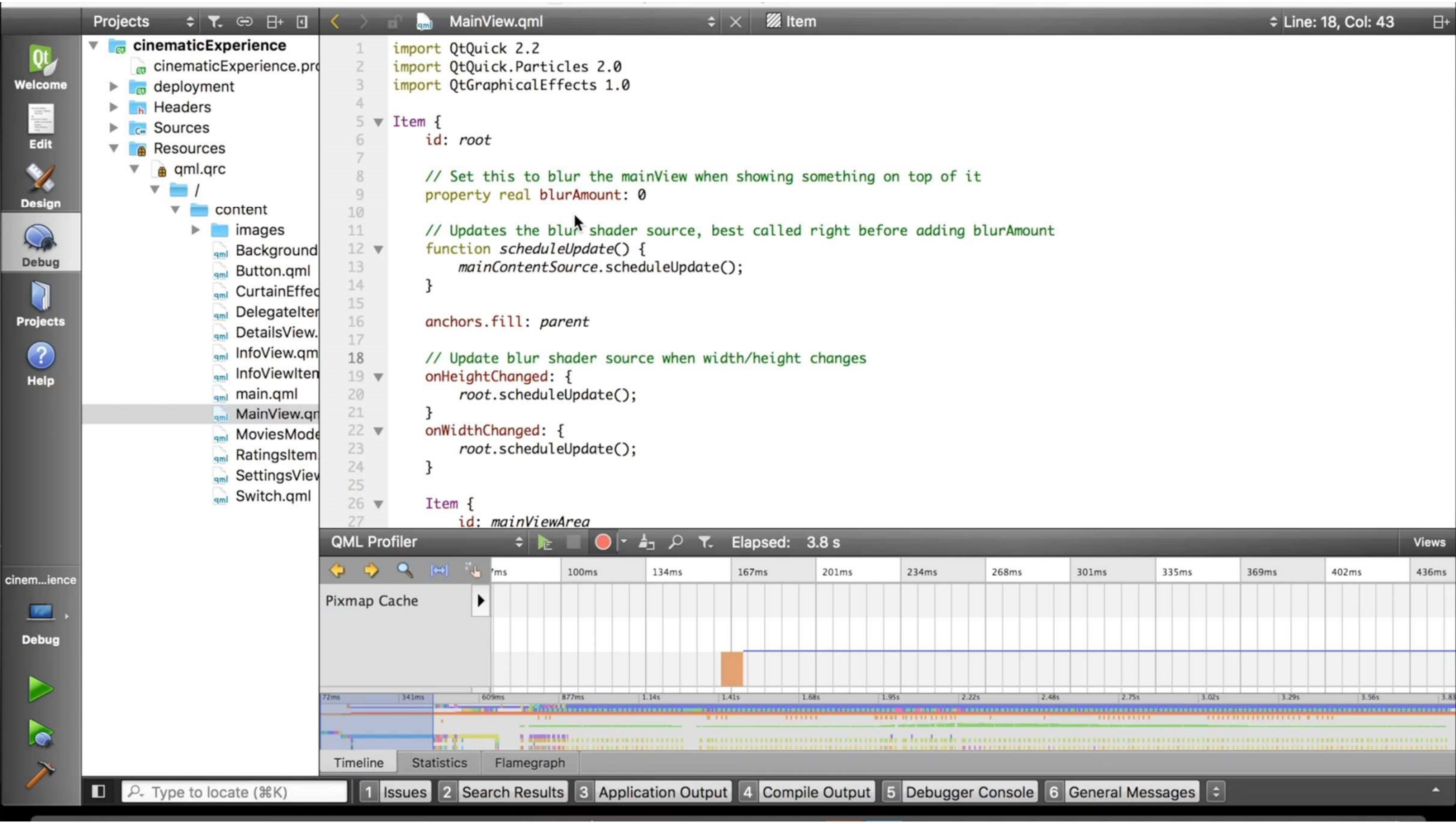
Qt Profiling 性能分析

利用QML Profiler进行应用软件的分析，降低不必要的性能损耗。





Memory
Allocated: 65536 bytes
Allocation: 1
Location: FastBlur.qml:366
Total: 1017120 bytes
Type: Heap Allocation


Projects
MainView.qml
Item
Line: 18, Col: 43
cinematicExperience
cinematicExperience.pro
deployment
Headers
Sources
Resources
qml.qrc
/
content
images
Background
Button.qml
CurtainEffec
DelegateIter
DetailsView.
InfoView.qm
InfoViewIten
main.qml
MainView.qr
MoviesMode
RatingsItem
SettingsVie
Switch.qml
import QtQuick 2.2
import QtQuick.Particles 2.0
import QtGraphicalEffects 1.0
Item {
id: root
// Set this to blur the mainView when showing something on top of it
property real blurAmount: 0
// Updates the blur shader source, best called right before adding blurAmount
function scheduleUpdate() {
mainContentSource.scheduleUpdate();
}
anchors.fill: parent
// Update blur shader source when width/height changes
onHeightChanged: {
root.scheduleUpdate();
}
onWidthChanged: {
root.scheduleUpdate();
}
Item {
id: mainViewArea
QML Profiler
Elapsed: 3.8 s
Views
100ms
134ms
167ms
201ms
234ms
268ms
301ms
335ms
369ms
402ms
436ms
Pixmap Cache
Timeline
Statistics
Flamegraph
Welcome
Edit
Design
Debug
Projects
Help
Debug
Type to locate (⌘K)
1 Issues
2 Search Results
3 Application Output
4 Compile Output
5 Debugger Console
6 General Messages

JavaScript Code 关于JavaScript的优化使用

- 属性绑定
  - QML优化引擎，简单的表达式不需要启动Javascript
  - 避免声明JavaScript中间变量
  - 避免在即时求值范围外访问属性（immediate evaluation scope：绑定表达式所在对象的属性，组件中的id，组件中的根元素id）
  - 需要用到属性进行运算时避免直接写操作。
- 属性解析
  - 避免频繁访问属性（在QML Profiler看到的调用频率比较高的部分，要尤其注意不要进行属性访问）

Qt Quick图片和布局优化

- 降低图片加载时间和内存开销
  - 异步加载
  - 设置图片尺寸

- 锚定布局
  - 在元素布局时，使用anchors锚布局比属性绑定效率更高
  - 坐标>锚定>绑定>JavaScript函数

元素生命周期设计

请使用Loader-----动态的加载和卸载一个组件

- 使用active属性，可以延迟实例化。
- 使用setSource()函数，提供初始属性值。
- asynchronous异步属性为true，在组件实例化时可提高流畅性。

渲染注意事项

- 避免使用Clip属性（默认禁用），剪切损失性能
- 被覆盖不可见的元素要设置他visible为false，通知引擎不绘制
- 透明与不透明-------不透明效率更高，全透明时请设置为不可见

使用Animation而不是Timer

- Qt优化了动画的实现，性能高于我们通过定时器触发属性的改变
- 传统方式使用Timer传统方式
- Timer触发动画性能低下，更耗电
- 结论：使用Animator或Animation元件

Bad

```
SvgImage {
    id: loadingIndicator

    Timer {
        id: loadingTimer
        interval: 250
        repeat: true
        onTriggered: loadingIndicator.rotation += 45
    }
```

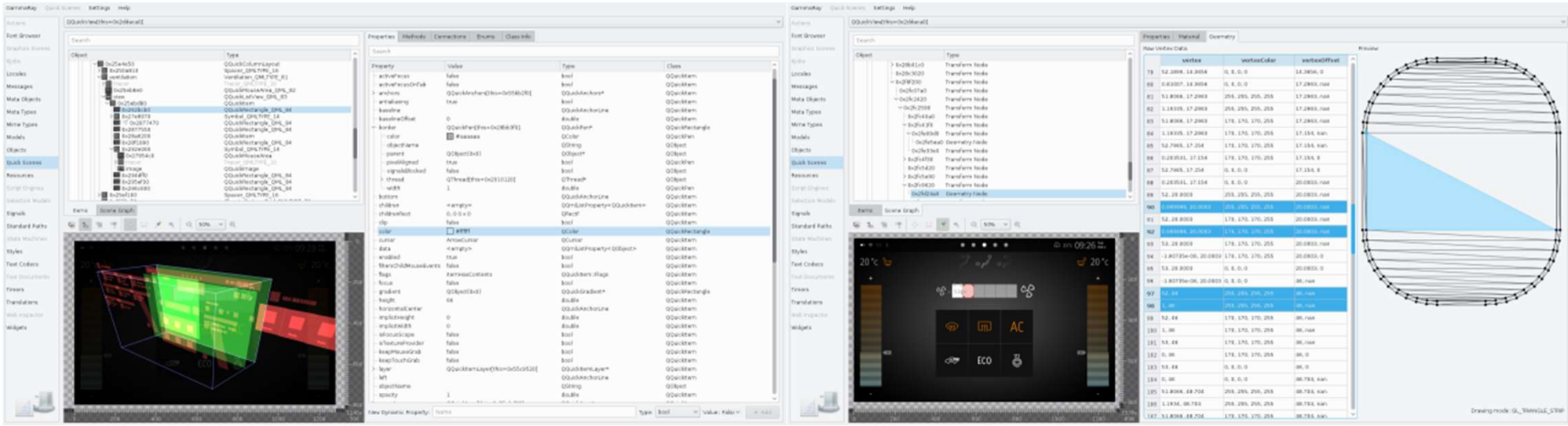
Good

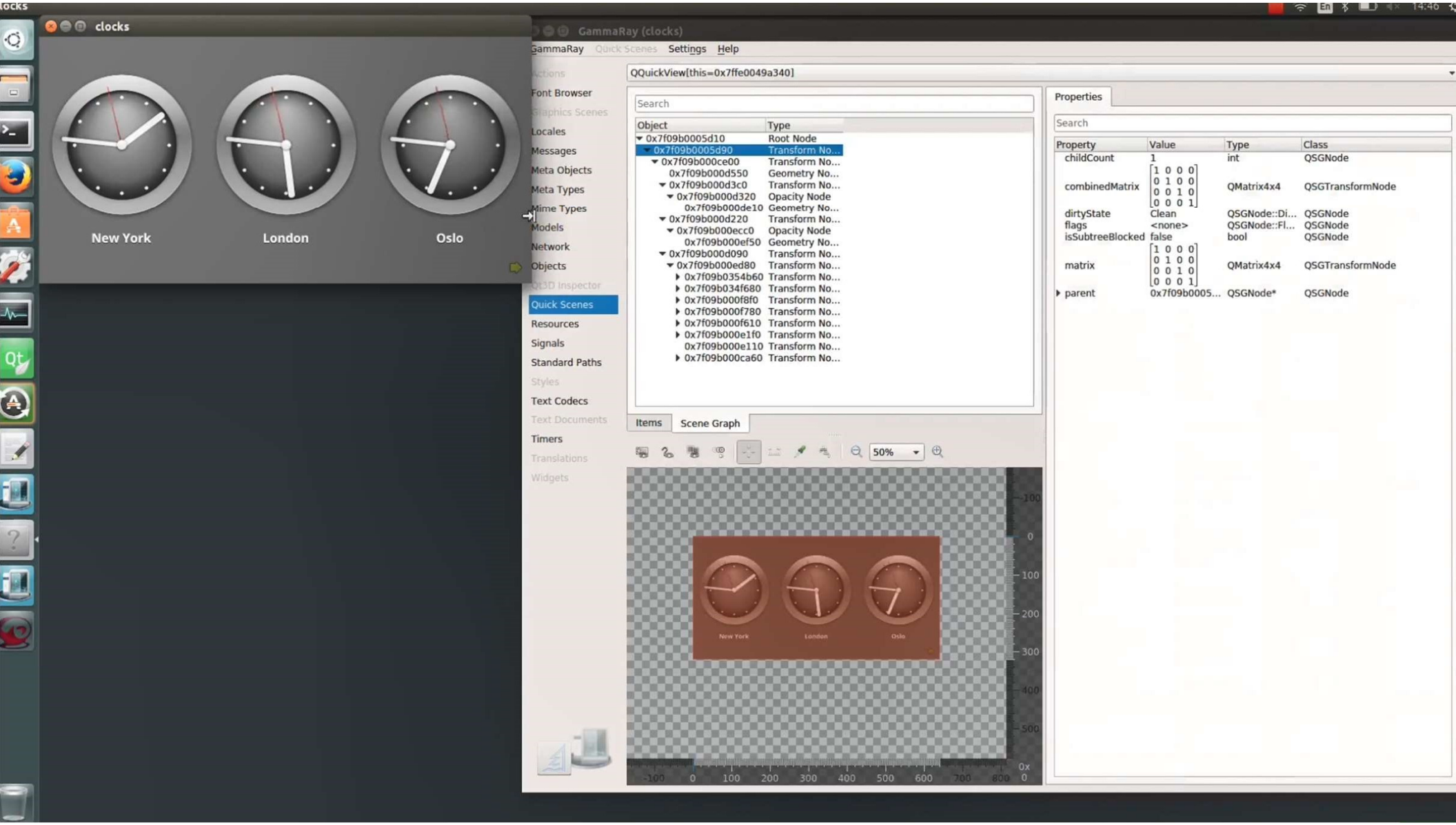
```
Image {
    id: control
    property bool running: false

    visible: running
    source: "../images/common/ico_caricamento_white.svg"
    NumberAnimation on rotation {
        running: control.running
        loops: Animation.Infinite
        duration: 2000
        from: 0 ; to: 360
    }
}
```

Qt Tool Chains- Gamma Ray

Qt Gamma Ray




clocks
New York
London
Oslo
GammaRay (clocks)
GammaRay
Quick Scenes
Settings
Help
Actions
Font Browser
Graphics Scenes
Locales
Messages
Meta Objects
Meta Types
Mime Types
Models
Network
Objects
Qt3D Inspector
Quick Scenes
Resources
Signals
Standard Paths
Styles
Text Codecs
Text Documents
Timers
Translations
Widgets
QQuickView[this=0x7ffe0049a340]
Search
Object
Type
0x7f09b0005d10 Root Node
0x7f09b0005d90 Transform No...
0x7f09b000ce00 Transform No...
0x7f09b000d550 Geometry No...
0x7f09b000d3c0 Transform No...
0x7f09b000d320 Opacity Node
0x7f09b000de10 Geometry No...
0x7f09b000d220 Transform No...
0x7f09b000ecc0 Opacity Node
0x7f09b000ef50 Geometry No...
0x7f09b000d090 Transform No...
0x7f09b000ed80 Transform No...
0x7f09b0354b60 Transform No...
0x7f09b034f680 Transform No...
0x7f09b000f8f0 Transform No...
0x7f09b000f780 Transform No...
0x7f09b000f610 Transform No...
0x7f09b000e1f0 Transform No...
0x7f09b000e110 Transform No...
0x7f09b000ca60 Transform No...
Items
Scene Graph
50%
Properties
Search
Property
Value
Type
Class
childCount 1 int QSGNode
combinedMatrix QMatrix4x4 QSGTransformNode
dirtyState Clean QSGNode::Di... QSGNode
flags <none> QSGNode::Fl... QSGNode
isSubtreeBlocked false bool QSGNode
matrix QMatrix4x4 QSGTransformNode
parent 0x7f09b0005... QSGNode* QSGNode

多线程编程

QThread

- QThread is the central class in Qt to run code in a different thread

- It's a QObject subclass
  - Not copiable/moveable
  - Has signals to notify when the thread starts/finishes
- It is meant to manage a thread

QThread Usage

- 为了创建新的线程执行相应处理，子类 QThread 并且重新实现 run()
- 实例化创建的线程子类，并调用 start()
- 想要设置线程优先级，通过设置 start()函数的*priority* 参数, 或者thread.setPriority(),默认继承所在线程的优先级。

```
QThread::IdlePriority
QThread::LowestPriority
QThread::LowPriority
QThread::NormalPriority
QThread::HighPriority
QThread::HighestPriority
QThread::TimeCriticalPriority
QThread::InheritPriority
```

QThread Usage

```
1  class MyThread : public QThread {
2  private:
3  void run() override {
4  // code to run in the new thread
5  }
6   };
```

```
1  MyThread *thread = new MyThread;
2  thread->start(); // starts a new thread which calls run()
3  // ...
4  thread->wait(); // waits for the thread to finish
```

QThread Usage

- 从run()函数返回后，线程会终止运行
- QThread::isRunning() 和 QThread::isFinished() 提供线程执行情况的信息
- 可以通过连接QThread::started() 和 QThread::finished() 信号获取相关状态
- 通过调用QThread::sleep() 函数临时停止线程的执行
  - Generally a *bad idea*, being event driven (or polling) is much much better----blockmode connect
- 通过调用wait()函数等待线程的结束。
  - 在调用terminate()后，由于不同操作系统的策略不同，可以通过wait等待线程的结束。
  - wait(unsigned long *time=ULONG_MAX*)
  - 通过参数设置等待的ms数

QThread 使用注意

在应用的非主线程中禁止:

- 执行任何GUI操作，例如：
  - 使用任何QWidget / Qt Quick / Qpixmap的API
  - 使用 QImage, Qpainter等等
- 调用 Q(Core|Gui)Application::exec()事件循环
- 阻塞GUI线程
- 在销毁相应的QThread对象之前，要销毁相应线程中的qobject

QThread 使用建议

你可以用这种方式:

- 在线程的run()里创建QObject
- 把QObject::deleteLater()槽函数连接到Qthread::finished()信号
- 将这个QObject移出到其他线程。

QThread 使用建议

```
 1  class MyThread : public QThread {
 2  private:
 3      void run() override {
 4          MyQObject obj1, obj2, obj3;
 5
 6          QScopedPointer<OtherQObject> p;
 7          if (condition)
 8              p.reset(new OtherQObject);
 9
10          auto anotherObj = new AnotherQObject;
11          connect(this, &QThread::finished,
12                  anotherObj, &QObject::deleteLater);
13
14          auto yetAnother = new YetAnotherQObject;
15
16          // ... do stuff ...
17
18          // Before quitting the thread, move this object to the main thread
19          yetAnother->moveToThread(qApp->thread());
20          // Somehow notify the main thread about this object,
21          // so it can be deleted there.
22          // Do not touch the object from this thread after this point!
23      }
24  };
```

QThread 的使用

› QThread的两种基本使用策略

- Without an event loop
- With an event loop

无事件循环（Event Loop）的QThread

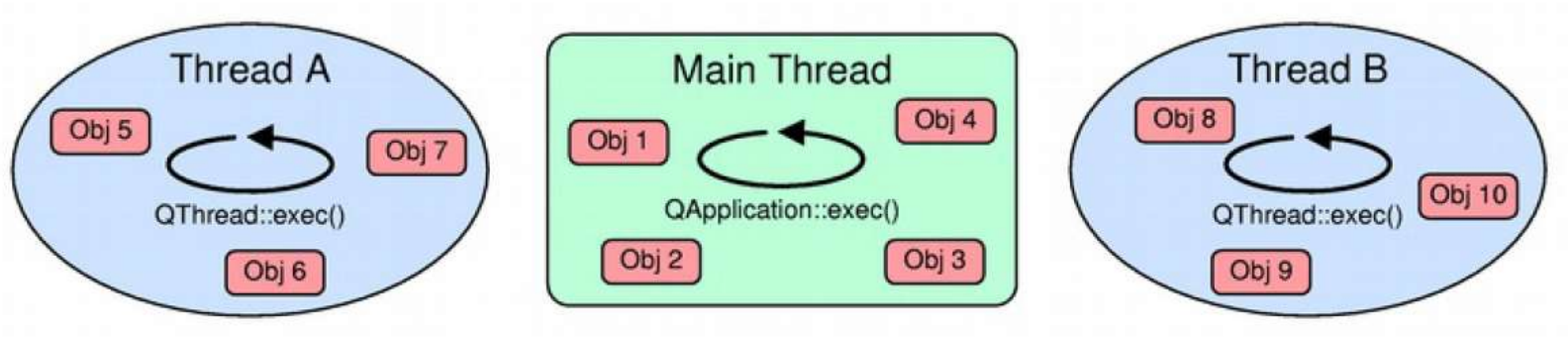
- Subclass QThread 并且override Qthread::run()函数。
- 创建实例并且通过Qthread::start()启动线程。

```
1  class MyThread : public QThread {
2  private:
3      void run() override {
4          loadFilesFromDisk();
5          doCalculations();
6          saveResults();
7      }
8  };
```

```
1  auto thread = new MyThread;
2  thread->start();
3  // some time later...
4  thread->wait();
```

有事件循环（Event Loop）的QThread

- 当使用timers, networking, *queuedConnections*等时，必须有事件循环。
- Qt支持独立线程的事件循环



- 每个线程的内部事件循环为线程内部的Qobject提供事件。

有事件循环（Event Loop）的QThread

› 在线程的run()函数里用QThread::exec() 开启事件循环:

```
1  class MyThread : public QThread {
2  private:
3      void run() override {
4          auto socket = new QTcpSocket;
5          socket->connectToHost(...);
6
7          exec(); // run the event loop
8
9          // cleanup
10     }
11 };
```

› QThread::quit() 或 QThread::exit() 退出事件循环。

有事件循环（Event Loop）的QThread

- Qthread::run()默认实现就是调用Qthread::exec()进入事件循环。
- 由于默认是进入Event loop，无需subclass Qthread，允许我们更简便的使用，例如：

```
1  auto thread = new QThread;
2
3  auto worker = new Worker;
4
5  connect(thread, &QThread::started, worker, &Worker::doWork);
6  connect(worker, &Worker::workDone, thread, &QThread::quit);
7
8  connect(thread, &QThread::finished, worker, &Worker::deleteLater);
9
10 worker->moveToThread(thread);
11 thread->start();
```

多线程要注意什么

对于共享资源的访问避免数据冲突

线程间同步（Synchronization）

› Qt提供跨平台的线程间同步的底层API

- QMutex提供了互斥量和互斥操作
- QSemaphore提供了一个整型信号灯（一种泛化的互斥）
- QWaitCondition is a condition variable
- QReadWriteLock提供了一个死锁允许同时进行读写操作
- QAtomicInt提供对整型的自动操作
- QAtomicPointer提供对指针的自动操作

关注Qt

Meet Qt北京站有奖提问问卷

微信公众号: Qt软件
WeChat ID: TheQtCompany

视频合集: Qt软件官方账号
Bilibili UID: 305085009


THE FUTURE
is written with
Qt