

[Qt创作者手册](#) > [分析 QML 应用程序](#)

分析 QML 应用程序

您可以使用 QML 性能分析器查找应用程序中典型性能问题的原因，例如运行缓慢、用户界面无响应、卡顿。典型的原因包括在太少的帧中执行过多的 JavaScript。所有 JavaScript 都必须返回，GUI 线程才能继续，如果 GUI 线程未准备好，帧将被延迟或丢弃。

类似性能问题的另一个典型原因是创建、绘制或更新不可见项目，这在 GUI 线程中需要时间。

触发长时间运行的 C++ 函数（如 paint 方法和信号处理程序）在 GUI 线程中也需要时间，但在 QML 性能分析器中更难看到，因为它不分析 C++ 代码。若要查找 JavaScript 的过度使用，请检查动画和场景图事件中的帧速率，查找间隙，并检查应用程序的行为是否符合预期。JavaScript 类别显示函数的运行时间，您应该尽量将其保持在每帧 16 毫秒以下。

若要查找由处理不可见项引起的问题，请查找丢弃的帧，并检查是否未使用过多的短绑定或按帧更新的信号处理程序。您还可以可视化场景图以检查场景布局并查找用户永远不可见的项目，因为它们位于屏幕外部或隐藏在其他可见元素下方。

如果即使 JavaScript 未运行，帧也会丢失，并且时间轴中存在较大的、无法解释的间隙，请检查您的自定义 `QQuickItem` 实现。您可以使用 `Valgrind` 或其他通用探查器来分析 C++ 代码。


您可以使用全栈跟踪从顶级 QML 或 JavaScript 向下跟踪到 C++，一直到内核空间。您可以在 [Chrome 跟踪格式查看器](#) 中查看收集的数据。

使用 QML 性能分析器

要在 QML 性能分析器中监视应用程序的性能，请执行以下操作：

1. 为了能够分析应用程序，必须为项目设置 QML 调试。有关更多信息，请参阅[设置 QML 调试](#)。
2. 在**项目**模式下，选择 Qt 版本为 4.7.4 或更高版本的[套件](#)。

注意：要在设备上分析应用程序，必须在[设备上](#)安装 Qt 库。

3. 选择“**分析**>**QML 分析器**”以分析当前应用程序。
4. 选择  (**开始**) 按钮以从 QML 性能分析器启动应用程序。

注意：如果数据收集未自动启动，请选择  “(禁用性能分析)”按钮。

在选择“**启用分析**”按钮之前，将收集数据。数据收集需要时间，因此，在显示数据之前可能会有延迟。

不要使用应用程序命令退出应用程序，因为当您选择“**启用性能分析**”按钮时，数据将发送到 QML 性能分析器。应用程序继续运行几秒钟，之后会自动停止。如果退出应用程序，则不会发送数据。

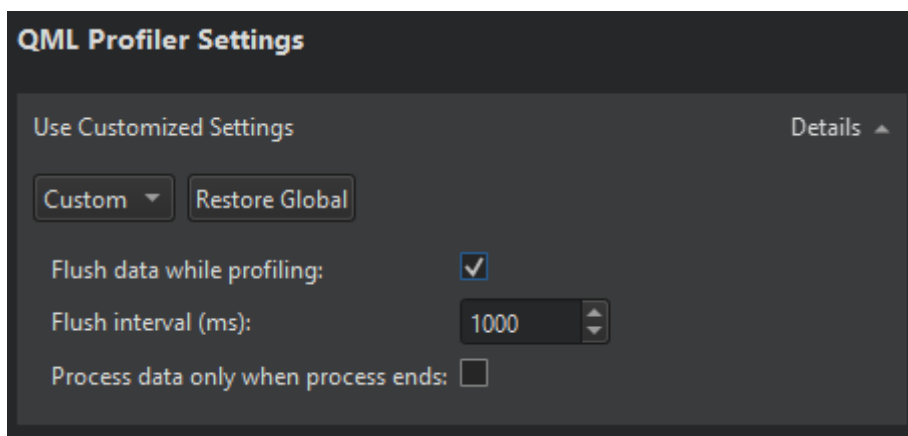
选择“**禁用分析**”按钮以禁用在启动应用程序时自动启动数据收集。再次选择该按钮时，数据收集将开始。

要保存所有收集的数据，请选择**分析>QML 分析器选项>保存 QML 跟踪**。要查看保存的数据，请选择**加载 QML 跟踪**。您还可以将保存的数据交付给其他人进行检查或加载他们保存的数据。

指定刷新设置

您可以为所有项目全局指定 QML 性能分析器的刷新设置，也可以为每个项目单独指定刷新设置。若要指定全局设置，请选择“**编辑>首选项>分析器**”。

要为特定项目指定自定义 QML 性能分析器设置，请选择“**项目>运行**”，然后在“**QML 性能分析器设置**”中选择“**自定义**”。若要还原项目的全局设置，请选择“**还原全局**”。



选中“分析时刷新数据”复选框可定期刷新数据，而不是在**分析停止时**刷新所有数据。这样可以节省目标设备上的内存，并缩短性能分析停止与显示数据之间的等待时间。

在 刷新间隔 字段中，以毫秒为单位设置**刷新间隔**。间隔越短，刷新数据的频率就越高。间隔越长，目标应用程序中需要缓冲的数据就越多，这可能会浪费内存。但是，刷新本身需要时间，这可能会扭曲分析结果。

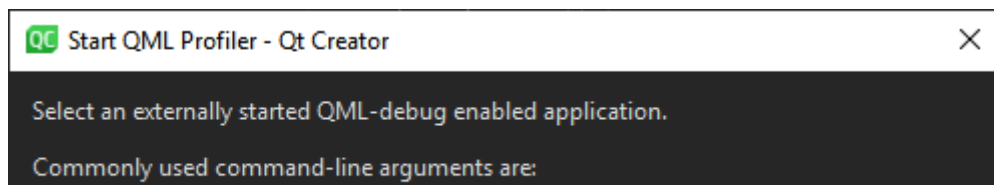
如果您有多个 QML 引擎，并且希望将所有引擎生成的数据聚合到一个跟踪中，请选中仅在**进程结束时处理数据**复选框。否则，当其中一个引擎停止时，性能分析将停止。

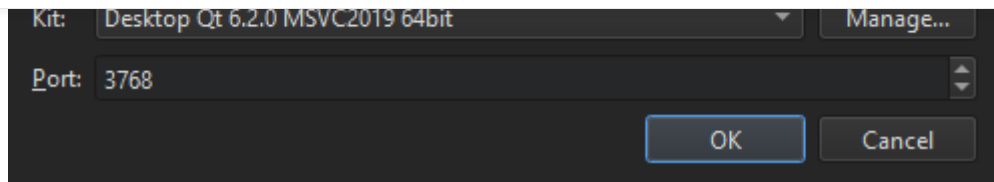
附加到正在运行的Qt快速应用程序

您可以分析不是由 Qt Creator 启动的 Qt Quick 应用程序。但是，您必须在项目构建设置中为应用程序启用 QML 调试和分析。有关更多信息，请参阅[设置 QML 调试](#)。

要附加到等待的应用程序，请执行以下操作：

1. 选择**分析>QML 分析器（附加到等待的应用程序）**。

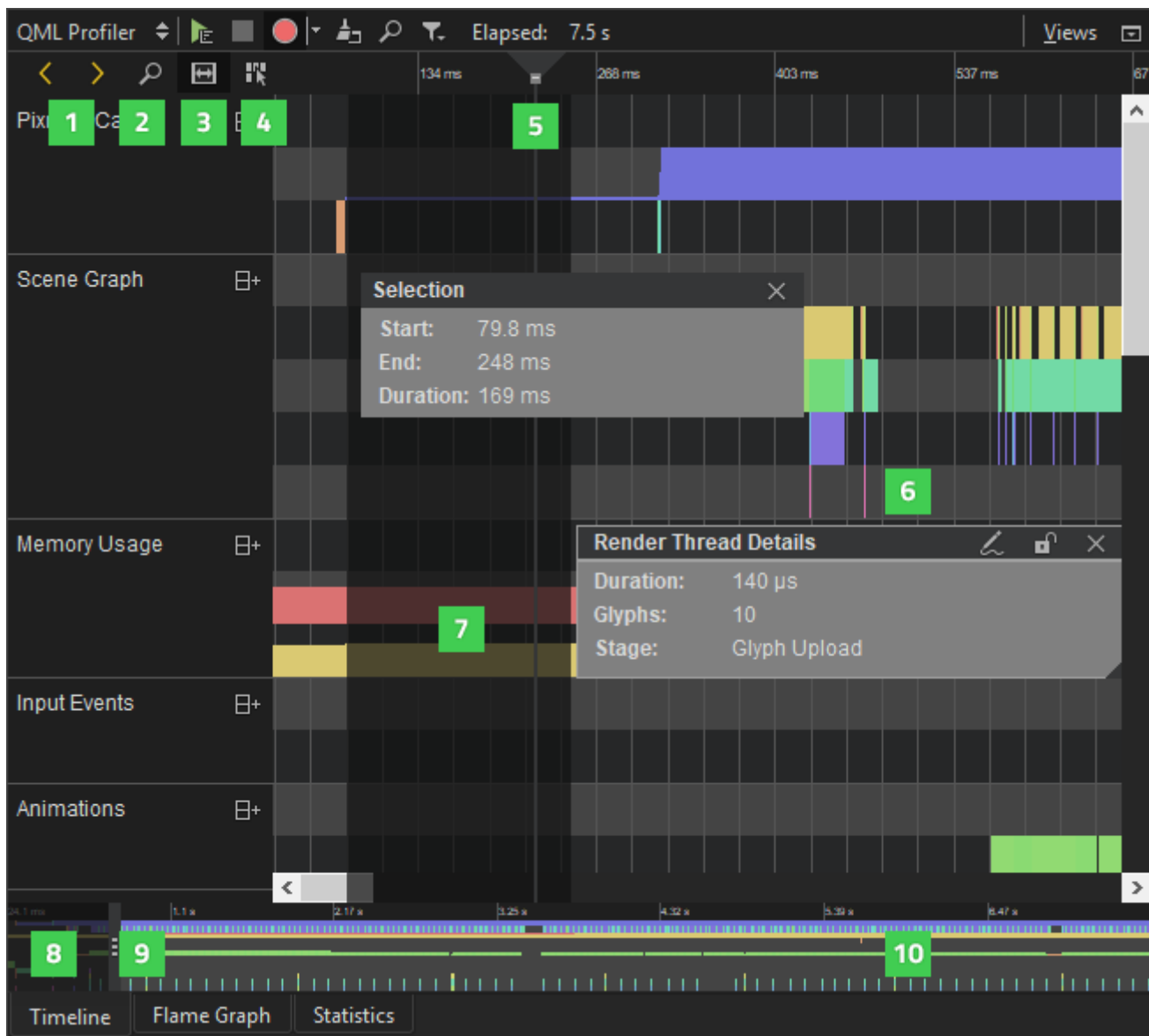




2. 在“工具包”中，选择用于生成应用程序的工具包。
3. 在“端口”中，指定要侦听的端口。
4. 选择“确定”。

分析收集的数据

时间轴视图显示 QML 和 JavaScript 执行的图形表示，以及所有记录事件的精简视图。



时间轴（6）中的每一行都描述了一种记录的 QML 事件类型。将光标移到行上的事件上，以查看它需要多长时间以及调用它在源中的位置。若要仅在选定事件时显示信息，请禁用“鼠标悬停时查看事件信息”按钮（4）。

大纲（10）总结了收集数据的时期。拖动缩放范围（8）或单击轮廓以在轮廓上移动。您还可以通过选择“跳转到上一个事件”和“跳转到下一个事件”按钮（1）在事件之间移动。

单击时间标尺以向时间轴添加垂直方向线（5）。

选择事件范围

您可以选择事件范围（7）以查看事件的帧速率，并将其与类似事件的帧速率进行比较。选择**“选择范围”**按钮（3）以激活选择工具。然后在时间轴中单击以指定事件范围的开始。拖动选择手柄以定义范围的末尾。范围的长度指示事件的帧速率。

您还可以使用事件范围来测量两个后续事件之间的延迟。在第一个事件的结束和第二个事件的开始之间放置一个范围。**“持续时间”**字段显示事件之间的延迟（以毫秒为单位）。

要放大事件范围，请双击它。

要在**“时间轴”**、**“统计”**和**“火焰图”**视图中缩小当前范围，请右键单击该范围，然后选择**“分析当前范围”**。要返回到完整范围，请在上下文菜单中选择**分析完整范围**。

若要删除事件范围，请关闭**“选择”**对话框。

了解数据

通常，时间轴视图中的事件指示 QML 或 JavaScript 执行所花费的时间。将鼠标移到它们上以查看详细信息。对于大多数事件，它们包括源代码中的位置、持续时间和源代码本身的一些相关部分。

可以单击某个事件，将代码编辑器中的光标移动到与该事件关联的代码部分。

以下类型的事件显示在时间线视图中的一行或多行上。事件类型的可用性取决于编译应用程序的 Qt 版本以及它使用的 Qt Quick 版本。

| 事件类别 | 描述 | 最低 Qt 版本 | Qt 快速版 |
|--------|--|------------|-------------------------|
| 像素图缓存 | 显示缓存的像素图数据的一般数量（以像素为单位）。此外，为正在加载的每张图片显示一个单独的事件，并提供有关其文件名和大小的详细信息。 | Qt 5.1 | Qt Quick 2 |
| 场景图 | 显示渲染场景图帧的时间以及为此而执行的各个阶段的一些附加计时信息。 | Qt 5.1 | Qt Quick 2 |
| 内存使用情况 | 显示 JavaScript 内存管理器的块分配。通常，内存管理器会将较大的内存块保留在一个整体中，然后以较小的位将它们分发给应用程序。如果应用程序请求的单个内存块超过特定大小，内存管理器将单独分配这些内存块。这两种操作模式显示为不同颜色的事件。第二行显示已分配内存的实际使用情况。这是应用程序实际请求的 JavaScript 堆量。 | Qt 5.4 | Qt Quick 2 |
| 输入事件 | 显示鼠标和键盘事件。 | Qt 4.7.4 | Qt Quick 1 或 Qt Quick 2 |
| 画 | 显示绘制每一帧场景所花费的时间。 | Qt 4.7.4 | Qt Quick 1 |
| 动画 | 显示处于活动状态的动画数量及其运行的帧速率。对于使用 Qt 5.3 或更高版本构建的应用程序，将显示有关渲染线程动画的信息，然后，渲染线程动画显示在单 | Qt 5.0 (Qt | Qt Quick |

| | | | |
|------------|---|---------------------|-------------------------|
| | | 4.7.4 | Quick 1 或 Qt Quick 2 |
| 创建 | 显示在场景中创建元素所花费的时间。在Qt Quick 2中，元素的创建分两个阶段进行。第一阶段是创建数据结构，包括子元素。第二阶段表示完成回调。但是，并非所有元素都会触发完成回调。这些阶段在时间轴中显示为单独的事件。对于使用5.2.1之前的Qt版本编译的Qt Quick 2应用程序，仅将顶级元素的创建显示为单个事件。 | Qt 4.7.4 (Qt 5.2.1) | Qt Quick 1 或 Qt Quick 2 |
| 捆绑 | 显示计算绑定的时间以及计算所需的时间。 | Qt 4.7.4 | Qt Quick 1 或 Qt Quick 2 |
| 处理信号 | 显示处理信号的时间以及处理所需的时间。 | Qt 4.7.4 | Qt Quick 1 或 Qt Quick 2 |
| JavaScript | 显示在绑定和信号处理程序后面执行实际 JavaScript 所花费的时间。它列出了可用于评估绑定或处理信号的所有 JavaScript 函数。 | Qt 5.3 | Qt Quick 2 |
| 快速3D | 显示渲染Qt Quick 3D帧所花费的时间、帧准备和同步的计时信息、粒子系统更新时间和粒子更新计数，以及纹理和网格内存分配和内存消耗。 | Qt 6.3 | Qt Quick 3D |

分析场景图事件

为了理解场景图类别，了解Qt Quick场景图的工作原理非常重要。有关详细说明，请参阅Qt [快速场景图](#)和Qt [快速场景图默认渲染器](#)。“**场景图**”类别中报告以下事件。并非所有事件都由所有呈现循环生成。在Windows和基本渲染循环中，所有内容都在同一线程中运行，GUI线程和渲染线程之间的区别毫无意义。

如果将环境变量设置为 QSG_RENDER_TIMING，则会从所分析的应用程序获得类似但时序略有不同的文本输出。为了便于定位，下面列出了差异。

| 事件类型 | 线程 | 渲染循环类型 | QSG_RENDER_TIMING 输出中的标签 | 描述 | 警告 |
|--------|--------|------------|--------------------------|--|---|
| 波兰语 | 图形用户界面 | 线程, 基本, 窗口 | 波兰语 | 在使用 <code>QQuickItem::updatePolish()</code> 渲染项目之前对项目进行最后润色。 | Qt 5.4 之前的 Qt 版本没有记录基本渲染循环的润色时间，而 Windows 渲染循环的润色时间不正确。 |
| 图形用户界面 | 图形渲染 | 螺旋锁 | | 执行连接到 <code>QQuickWindow::afterAnimating()</code> 信号的插槽时 | 没有 |

| | | | | | |
|------------|--------|----------|---------------|--|---|
| | 界面 | | | 在这渲染线程同步之前很久就开始了，那么 GUI 线程中就有空闲时间，您可以使用它来运行额外的 QML 或 JavaScript。 | |
| 图形用户界面线程同步 | 图形用户界面 | 螺纹 | 已阻止同步 | GUI 线程被阻塞，等待呈现线程的时间。 | 没有 |
| 动画 | 图形用户界面 | 线程，窗口 | 动画 | 在 GUI 线程中推进动画。基本渲染循环不会驱动动画与渲染同步。这就是使用基本渲染循环时不显示动画事件的原因。观看“动画”类别以查看本例中的 动画 计时。 | 没有 |
| 渲染线程同步 | 呈现 | 线程，基本，窗口 | 帧渲染...同步 | 使用 <code>QQuickItem::updatePaintNode()</code> 将 QML 状态同步到场景中。 | 没有 |
| 呈现 | 呈现 | 线程，基本，窗口 | 帧渲染...呈现 | 渲染帧所花费的总时间，包括准备所有必要的数据并将其上传到 GPU。这是总渲染时间。不要将其与下面的 净渲染渲染 时间混淆。 | 对于 Qt 5.5 之前的 Qt 版本，由于使用不同的不同步计时器来测量它们，总渲染时间和渲染时间的以下分解可能会错位几微秒。例如， 渲染预处理 似乎在 渲染线程同步 完成之前开始。 |
| 交换 | 呈现 | 线程，基本，窗口 | 帧渲染...交换 | 渲染后交换帧。 | 设置 <code>QSG_RENDER_TIMING</code> 触发的交换时间输出对于基本渲染循环和 Qt 5.4 之前版本的 Qt 不正确。QML 探查器显示正确的交换时间。 |
| 渲染预处理 | 呈现 | 线程，基本，窗口 | 渲染器中的时间...预处理 | 在所有需要预处理的节点上调用 <code>QSGNode::preprocess()</code> 。这是总 渲染 步骤的一部分。 | 可能无法与 Qt 5.5 之前的 Qt 版本正确对齐。 |
| 渲染更新 | 呈现 | 线程，基本，窗口 | 渲染器中的时间...更新 | 迭代和处理场景中的所有节点，以更新其几何体、变换、不透明度和其他状态。在 渲染线程同步 阶段，每个节点都使用 GUI 线程的状态单独更新。在 渲染更新 中，所有节点组合在一起以创建最终场景。这是总 渲染 步骤的一部分。 | 可能无法与 Qt 5.5 之前的 Qt 版本正确对齐。 |
| 呈现绑定 | 呈现 | 线程 | 渲染器中的时间...捆绑 | 为 OpenGL 渲染绑定正确的帧缓冲。这是总 渲染 步骤的一部分。 | 可能无法与 Qt 5.5 之前的 Qt 版本正确对齐。 |

| | | | | | |
|------|----|------------|--------------|---------------------------------------|-------------------------------|
| 渲染渲染 | 呈现 | 线程, 基本, 窗口 | 渲染器中的时间...渲染 | 通过OpenGL将所有数据发送到GPU的实际过程。这是总渲染步骤的一部分。 | 可能无法与Qt 5.5 之前的 Qt 版本正确对齐。 |
| 材料编译 | 呈现 | 线程, 基本, 窗口 | 着色器已编译 | 编译 GLSL 着色器程序。 | 没有 |
| 字形渲染 | 呈现 | 线程, 基本, 窗口 | 符号。。。渲染 | 将字体字形呈现为纹理。 | Qt 5.4 之前的 Qt 版本报告这些事件的时间不正确。 |
| 字形上传 | 呈现 | 线程, 基本, 窗口 | 符号。。。上传 | 将字形纹理上传到 GPU。 | Qt 5.4 之前的 Qt 版本报告这些事件的时间不正确。 |
| 纹理绑定 | 呈现 | 线程, 基本, 窗口 | 素色质地...捆 | 使用 glBindTextures 在 OpenGL 上下文中绑定纹理。 | 没有 |
| 纹理转换 | 呈现 | 线程, 基本, 窗口 | 素色质地...转换 | 转换格式并缩小图像以使其适合作为纹理。 | 没有 |
| 纹理旋转 | 呈现 | 线程, 基本, 窗口 | 素色质地...旋转 | 如有必要, 在 CPU 上滑动纹理数据。 | 没有 |

Qt创建者手册8.0.2

Topics >

| | | | | | |
|-----------|----|------------|-------------|-------------------|----|
| 纹理上传 | 呈现 | 线程, 基本, 窗口 | 纹理上传 | | |
| 纹理 Mipmap | 呈现 | 线程, 基本, 窗口 | 素色质地...米普地图 | 在 GPU 上对纹理进行微调映射。 | 没有 |
| 纹理删除 | 呈现 | 线程, 基本, 窗口 | 删除了纯纹理 | 从 GPU 中删除不必要的纹理。 | 没有 |

分析Qt快速3D事件

以下是Qt Quick 3D的事件列表。每个渲染帧都包含同步、准备和渲染阶段，这些阶段按该顺序完成。同步发生在场景图同步阶段，而准备和渲染发生在场景图渲染阶段。

设置环境变量还将提供不同渲染通道的渲染调用计数的额外文本输出。这些调用计数在呈现帧事件中汇总。
QSG_RENDERER_DEBUG=render

| 事件类型 | 线 | 描述 |
|---------|--------|---|
| 渲染帧 | 呈现 | 帧的呈现时间。还显示绘制调用数。 |
| 准备帧 | 呈现 | 准备帧所花费的时间。资源在准备阶段分配和加载。场景加载后的第一帧通常比其他帧花费更长的时间，因为大多数资源都是在那时加载的。 |
| 同步帧 | 呈现 | 同步帧的时间。同步负责从前端更新后端值。还管理Qt Quick Scene Graph和Qt Quick 3D之间的共享资源。 |
| 网格负载 | 呈现 | 网格的加载时间。显示所有网格的总内存使用量。还显示卸载。 |
| 自定义网格加载 | 呈现 | 自定义网格的加载时间。显示所有网格的总内存使用量。还显示卸载。 |
| 纹理加载 | 呈现 | 纹理的加载时间。显示所有纹理的总内存使用量。还显示卸载。 |
| 生成着色器 | 呈现 | 为材质生成着色器的时间。 |
| 加载着色器 | 呈现 | 加载内置着色器的时间。 |
| 粒子更新 | 图形用户界面 | 粒子系统的更新时间。显示更新的粒子数。 |
| 网格内存消耗 | 呈现 | 显示网格内存总消耗的条形图。 |
| 纹理内存消耗 | 呈现 | 显示总纹理内存消耗的条形图。 |

查看统计信息

“统计信息”视图显示触发每个绑定、创建、编译、JavaScript 或信号事件的次数以及平均花费的时间。这允许您检查需要优化的事件。大量出现可能表示事件被不必要地触发。若要查看发生的中位数、最长和最短时间，请在上下文菜单中选择“扩展事件统计信息”。

单击某个事件以在代码编辑器的源代码中移动到该事件。

| Location | Type | Time in Percent ▾ | Total Time | Self Time in Percent | Self Time | Calls | Mean Time | Details |
|---------------|--------------|-------------------|------------|----------------------|-----------|-------|-----------|-----------------|
| <program> | | 100 % | 304 ms | 0.00 % | 0 ns | 1 | 304 ms | Main program |
| clocks.qml:59 | Creating | 78.51 % | 238 ms | 0.15 % | 466 μs | 2 | 119 ms | QtQuick/ListV |
| Clock.qml:141 | Creating | 77.87 % | 236 ms | 77.87 % | 236 ms | 16 | 14.8 ms | QtQuick/Text |
| clocks.qml:0 | Compiling | 14.19 % | 43.1 ms | 12.63 % | 38.3 ms | 1 | 43.1 ms | clocks.qml |
| Clock.qml:87 | Handling ... | 4.28 % | 13 ms | 0.42 % | 1.28 ms | 553 | 23.5 μs | onTriggered: c |
| Clock.qml:87 | JavaScript | 3.86 % | 11.7 ms | 0.70 % | 2.13 ms | 553 | 21.2 μs | expression for |
| Clock.qml:77 | JavaScript | 3.16 % | 9.58 ms | 2.21 % | 6.72 ms | 553 | 17.3 μs | timeChanged |
| clocks.qml:90 | Creating | 2.41 % | 7.3 ms | 2.41 % | 7.3 ms | 2 | 3.65 ms | QtQuick/Imag |
| Clock.qml:0 | Compiling | 1.56 % | 4.73 ms | 1.56 % | 4.73 ms | 1 | 4.73 ms | Clock.qml |
| Clock.qml:130 | Binding | 0.64 % | 1.96 ms | 0.60 % | 1.81 ms | 72 | 27.2 μs | angle: clock.se |
| clocks.qml:67 | Creating | 0.32 % | 978 μs | 0.02 % | 65.1 μs | 8 | 122 μs | Clock.qml |

| Caller | Type | Total Time | Calls | Caller Description | Callee | Type | Total Time | Calls | Callee Description |
|---------------|----------|------------|-------|--------------------|---------------|----------|------------|-------|--------------------|
| <program> | | 238 ms | 1 | Main Program | Clock.qml:141 | Creating | 235 ms | 4 | QtQuick/Text |
| clocks.qml:54 | Creating | 224 µs | 1 | QtQuick/Rectangle | Clock.qml:95 | Creating | 671 µs | 4 | QtQuick/Image |
| | | | | | Clock.qml:94 | Creating | 550 µs | 4 | QtQuick/Image |
| | | | | | clocks.qml:67 | Creating | 464 µs | 4 | Clock.qml |
| | | | | | Clock.qml:137 | Creating | 147 µs | 4 | QtQuick/Image |

TimelineFlame GraphStatistics

“**呼叫方**”和“**呼叫方**”窗格显示事件之间的依赖关系。它们允许您检查应用程序的内部功能。“**调用方**”窗格汇总了触发绑定的 QML 事件。这会告诉您导致绑定更改的原因。“**调用**”窗格汇总了绑定触发的 QML 事件。这会告诉您在更改绑定时哪些 QML 事件会受到影响。

单击某个事件以在代码编辑器的源代码中移动到该事件。

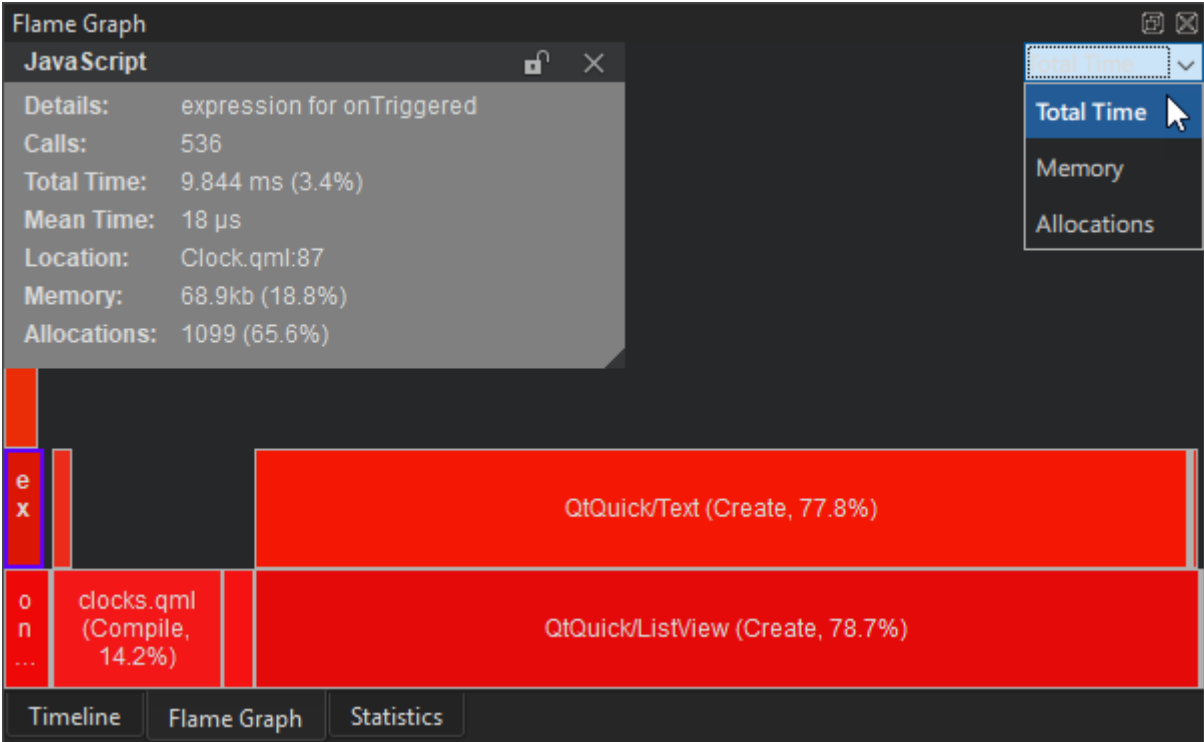
在“**时间轴**”视图中选择事件时，有关该事件的信息将显示在“**统计信息**”和“**火焰图**”视图中。

若要将一个视图或行的内容复制到剪贴板，请在上下文菜单中选择“**复制表**”或“**复制行**”。

JavaScript 事件仅在**使用**Qt Quick 2并使用Qt 5.3或更高版本编译的应用程序的统计信息视图中显示。

将统计数据可视化为火焰图

Flame Graph视图显示了 QML 和 JavaScript 执行的更简洁的统计概述。在“**总时间**”视图中，水平条显示相对于所有 JavaScript 和 QML 事件的总运行时间，对某个函数的所有调用加起来的时间量。嵌套显示哪些函数被哪些其他函数调用。



要查看函数分配的内存总量，请在下拉菜单中选择**内存**。

若要查看函数执行的内存分配数，请选择“**分配**”。

双击视图中的某个项目可将其放大。双击视图中的空白区域可再次缩小。

与**时间轴**视图不同，**火焰图**视图不显示根本没有运行 QML 或 JavaScript 的时间跨度。因此，它不适合分析每帧执行时间。但是，很容易看到各种 QML 和 JavaScript 事件的总体影响。

© 2022 Qt有限公司 此处包含的文档贡献的版权归他们各自的所有者。此处提供的文档根据自由软件基金会发布的GNU自由文档许可证版本 1.3的条款进行许可。Qt和相应的徽标是Qt有限公司在芬兰和/或全球其他国家的商标。所有其他商标均为财产 其各自所有者。



联系我们

公司

- 关于我们
- 投资者
- 编辑部
- 职业
- 办公地点

发牌

- 条款和条件
- 开源
- 常见问题

支持

- 支持服务
- 专业服务
- 合作伙伴
- 训练

对于客户

- 支持中心
- 下载
- Qt登录
- 联系我们
- 客户成功案例

社区

- 为Qt做贡献
- 论坛
- 维基
- 下载
- 市场