搜索

可视化 Chrome 跟踪事件

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。这使您能够测量应用程序的性能，并检查它是否受 CPU 或 I/O 限制或受同一系统上运行的其他应用程序的影响。通过跟踪，可以深入了解系统正在执行的操作以及应用程序以特定方式执行的原因。它指示如何使用硬件以及内核和应用程序正在执行的操作。

跟踪信息还可以让您更深入地了解QML 分析器收集的数据。例如，您可以检查为什么一个微不足道的绑定评估需要这么长时间。这可能是由于正在执行C++或磁盘 I/O 速度缓慢引起的。

多个跟踪工具（例如）可以生成有关 Chrome 跟踪事件的信息（例如）。您可以在Qt Creator中打开CTF文件进行查看。这在查看大于 100 MB 的跟踪文件时特别有用，由于其内存使用率高，使用内置跟踪查看器 () 很难查看这些文件。`chrome://aboutchrome://tracing`

可视化工具支持LTTng跟踪框架生成的转换为 CTF 的数据中使用的所有事件类型。但是，不支持某些更高级的事件类型（例如，在 Android 系统跟踪中使用的事件类型）。可视化工具以静默方式忽略不受支持的事件类型。

可视化工具支持以下事件类型：

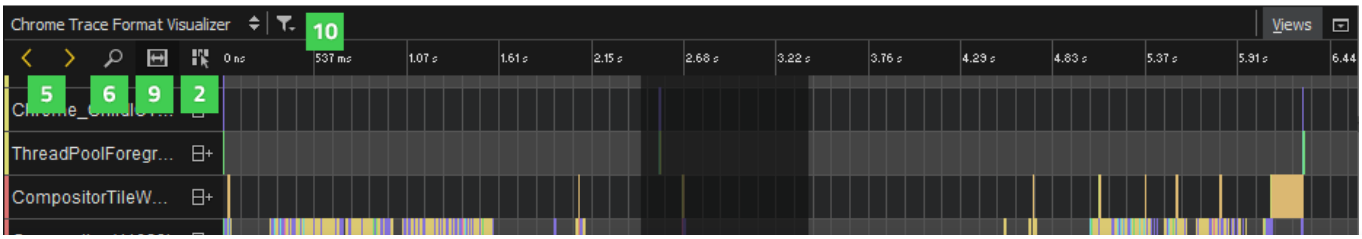
- 开始、结束、持续时间和即时事件
- 计数器事件（图形）
- 元数据事件（进程和线程名称）

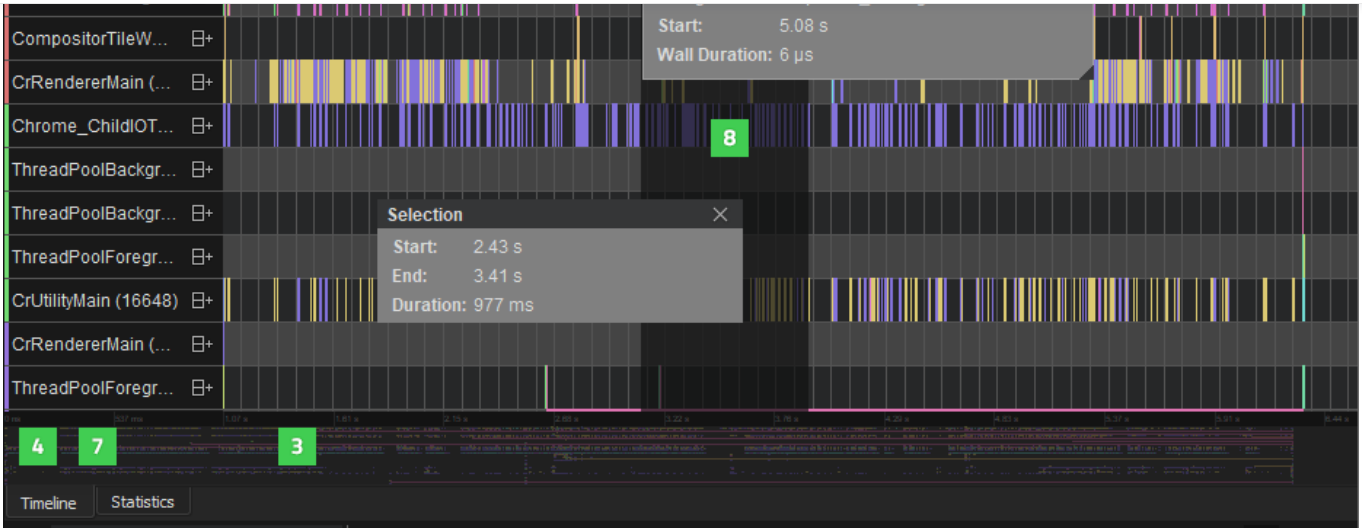
打开 JSON 文件

要打开 JSON 文件进行查看，请选择**分析**>**Chrome 跟踪格式查看器**>**加载 JSON 文件**。

可视化事件

“**时间线**”视图显示跟踪事件的图形表示形式和所有记录事件的精简视图。





时间轴中的每个类别描述应用程序中的一个线程。将光标移到行上的事件 （1） 上以查看其持续时间和事件类别。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 （2）。

大纲（3）总结了收集数据的时期。拖动缩放范围 （4） 或单击轮廓以在轮廓上移动。要在事件之间移动，请选择"跳转到上一个事件"和**"跳转到下一个事件"**按钮 （5）。

选择**"显示缩放滑块"**按钮 （6） 以打开可用于设置缩放级别的滑块。您也可以拖动缩放手柄 （7）。要重置默认缩放级别，请右键单击时间轴以打开上下文菜单，然后选择**重置缩放**。

选择（**限制为线程**）按钮 （10） 以选择要显示的线程。

选择事件范围

您可以选择事件范围 （8） 以查看其表示的时间或放大跟踪的特定区域。选择**"选择范围"**按钮 （9） 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。

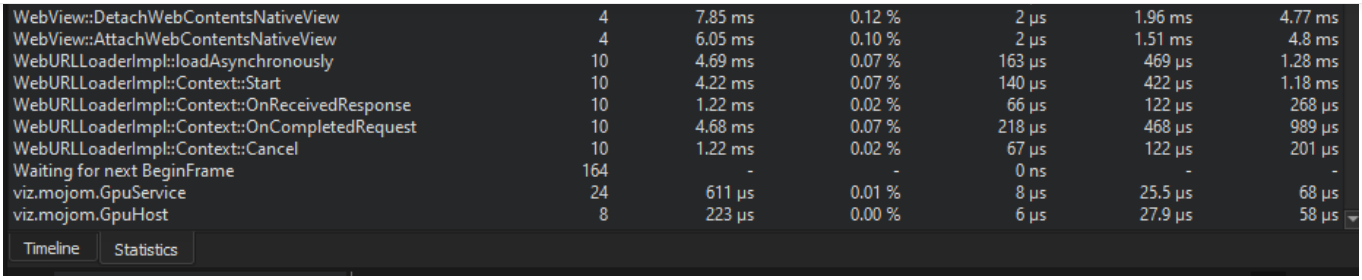
您还可以使用事件范围来测量两个后续事件之间的延迟。在第一个事件的结束和第二个事件的开始之间放置一个范围。**"持续时间"**字段显示事件之间的延迟（以毫秒为单位）。

要放大事件范围，请双击它。

若要删除事件范围，请关闭**"选择"**对话框。

查看统计信息

Title	Count	Total Time	Percentage	Minimum Time	Average Time	Maximum Time
XHRReadyStateChange	39	4.49 ms	0.07 %	7 µs	115 µs	307 µs
XHRLoad	10	174 µs	0.00 %	6 µs	17.4 µs	30 µs
WorkerThreadThread born	1	-	-	0 ns	-	-
WorkerThreadThread active	502	23.1 s	368.26 %	29 µs	46.1 ms	5.37 s
WindowTreeHost::OnHostMovedInPixels	1	126 µs	0.00 %	126 µs	126 µs	126 µs
WidgetMsg_WasShown	2	278 µs	0.00 %	119 µs	139 µs	159 µs
WidgetMsg_WasHidden	2	383 µs	0.01 %	67 µs	192 µs	316 µs
WidgetMsg_UpdateVisualProperties	1	56 µs	0.00 %	56 µs	56 µs	56 µs
WidgetMsg_UpdateScreenRects	4	114 µs	0.00 %	13 µs	28.5 µs	44 µs
WidgetMsg_SetActive	4	239 µs	0.00 %	4 µs	59.8 µs	221 µs
WidgetInputHandlerManager::HandledInputEvent	212	4.32 ms	0.07 %	3 µs	20.4 µs	152 µs
WidgetInputHandlerManager::DidHandleInputEventAndOverscroll	212	7.36 ms	0.12 %	10 µs	34.7 µs	121 µs
WidgetInputHandlerImpl::DispatchEvent	212	14.5 ms	0.23 %	19 µs	68.6 µs	200 µs
WidgetHostMsg_UpdateScreenRects_ACK	4	116 µs	0.00 %	14 µs	29 µs	44 µs
WidgetHostMsg_TextInputStateChanged	2	8 µs	0.00 %	3 µs	4 µs	5 µs
WidgetHostMsg_SetCursor	3	920 µs	0.01 %	98 µs	307 µs	552 µs
WidgetHostMsg_SelectionBoundsChanged	1	26 µs	0.00 %	26 µs	26 µs	26 µs
WidgetHostMsg_FrameSwapMessages	1	7 µs	0.00 %	7 µs	7 µs	7 µs
WidgetHostMsg_DidFirstVisuallyNonEmptyPaint	1	15 µs	0.00 %	15 µs	15 µs	15 µs



WebView::DetachWebContentsNativeView	4	7.85 ms	0.12 %	2 µs	1.96 ms	4.77 ms
WebView::AttachWebContentsNativeView	4	6.05 ms	0.10 %	2 µs	1.51 ms	4.8 ms
WebURLLoaderImpl::loadAsynchronously	10	4.69 ms	0.07 %	163 µs	469 µs	1.28 ms
WebURLLoaderImpl::Context::Start	10	4.22 ms	0.07 %	140 µs	422 µs	1.18 ms
WebURLLoaderImpl::Context::OnReceivedResponse	10	1.22 ms	0.02 %	66 µs	122 µs	268 µs
WebURLLoaderImpl::Context::OnCompletedRequest	10	4.68 ms	0.07 %	218 µs	468 µs	989 µs
WebURLLoaderImpl::Context::Cancel	10	1.22 ms	0.02 %	67 µs	122 µs	201 µs
Waiting for next BeginFrame	164	-	-	0 ns	-	-
viz.mojom.GpuService	24	611 µs	0.01 %	8 µs	25.5 µs	68 µs
viz.mojom.GpuHost	8	223 µs	0.00 %	6 µs	27.9 µs	58 µs

Timeline Statistics

"**统计信息**"视图显示时间轴中每个函数包含的样本数、总数以及何时位于堆栈顶部（调用）。这使您可以检查需要优化哪些功能。大量出现可能表示函数被不必要地触发或需要很长时间才能执行。`self`

收集 LTTng 数据

LTTng 是一个适用于 Linux 的跟踪工具包，您可以将其应用于嵌入式 Linux 系统，以了解如何优化应用程序的启动时间。

从Qt 5.13开始，Qt为自定义用户空间跟踪点提供了一组内核跟踪点和一个跟踪子系统。

配置内核

要使用 LTTng，您必须在构建内核之前为内核设置以下配置选项：

- `CONFIG_HIGH_RES_TIMERS`
- `CONFIG_KALLSYMS`
- `CONFIG_MODULES`
- `CONFIG_TRACEPOINTS`

我们建议您设置以下附加选项：

- `CONFIG_EVENT_TRACING`
- `CONFIG_HAVE_SYSCALL_TRACEPOINTS`
- `CONFIG_KALLSYMS_ALL`

在Yocto中，您可以在**菜单**>**配置**>**内核黑客**>**跟踪器**中激活上述选项。

安装 LTTng

构建内核并将其部署到设备上后，需要在设备上安装以下 LTTng 软件包：

- `lttng-tools`控制跟踪会话
- `lttng-modules`对于内核跟踪点
- `lttng-ust`对于用户空间跟踪点

在Yocto中，您只需要启用即可。`EXTRA_IMAGE_FEATURES += "tools profile"`

使用跟踪点构建Qt

跟踪点不断添加到Qt版本中。要使用它们，您需要使用选项自己构建Qt。`configure -trace lttng`

最后，你调用开始跟踪。`lttng createlttng enable-channel kernel -kkernel_eventslttng enable-eventlttng start`

你打电话停止跟踪。您可以使用来设置会话的长度。停止后，您可以调用以销毁会话。`lttng stopsleeplttng destroy`

您可以编写并运行包含上述命令的脚本来启动和停止全栈跟踪。您可以使用 来执行脚本。`systemd`

启用跟踪点

根据您在 LTTng 会话中启用的跟踪点记录数据。通常，启用调度程序切换、系统调用和 Qt 跟踪点很有用。

调度程序交换机跟踪点

当应用程序因抢占而切换时，例如，当另一个进程有机会在 CPU 内核上运行时，将达到调度程序切换跟踪点。启用调度程序 schwitch 跟踪点以记录当前正在运行的线程及其所属的进程，以及进程启动和停止的时间。

系统调用跟踪点

系统调用跟踪点可帮助您了解发生调度程序切换的原因。以下是要跟踪的系统调用的示例：

- `openat`并将文件描述符映射到文件名`close`
- `mmap`将页面错误映射到文件
- `read`并且由 I/O 操作触发`write`
- `nanosleep`，，并解释调度程序开关`futexpoll`
- `ioctl`控制 GPU 和显示器

将 LTTng 数据转换为 CTF

ctf2ctf工具用于解析二进制通用跟踪格式 （CTF） 并将其转换为 Chrome 跟踪格式 （CTF）。它执行以下自定义任务，使录制文件更易于阅读：`babeltrace`

- 将文件描述符映射到文件名
- 将页面错误映射到文件名
- 用名称注释中断并阻止设备
- 将 UTF-16QString数据转换为 UTF-8 字符串
- 计算内存页分配

要生成包含 Chrome 跟踪格式跟踪数据的 JSON 文件，请在命令行中输入以下命令：

```
ctf2ctf -o trace.json path/to/lttng trace/
```

‹ 使用 Cppcheck 分析代码 运行自动测试 ›

The Qt Company

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

反馈 登录