可视化浏览器跟踪事件

您可以使用*完整堆栈跟踪*来跟踪从顶级 QML 或 JavaScript 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。它指示硬件的利用方式以及内核和应用程序正在执行的操作。

跟踪信息还可以为您提供对 QML 探查器收集的数据的其他见解。例如，您可以检查为什么一个简单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 速度慢造成的。

多个跟踪工具（例如 ）可以生成有关 Chrome 跟踪格式 （CTF） 的 Chrome 跟踪事件的信息。您可以在Qt创建器中打开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，由于内存使用率高，因此很难使用内置跟踪查看器 （） 查看这些文件。`chrome://aboutchrome://tracing`

可视化工具支持 LTTng 跟踪框架生成并转换为 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：

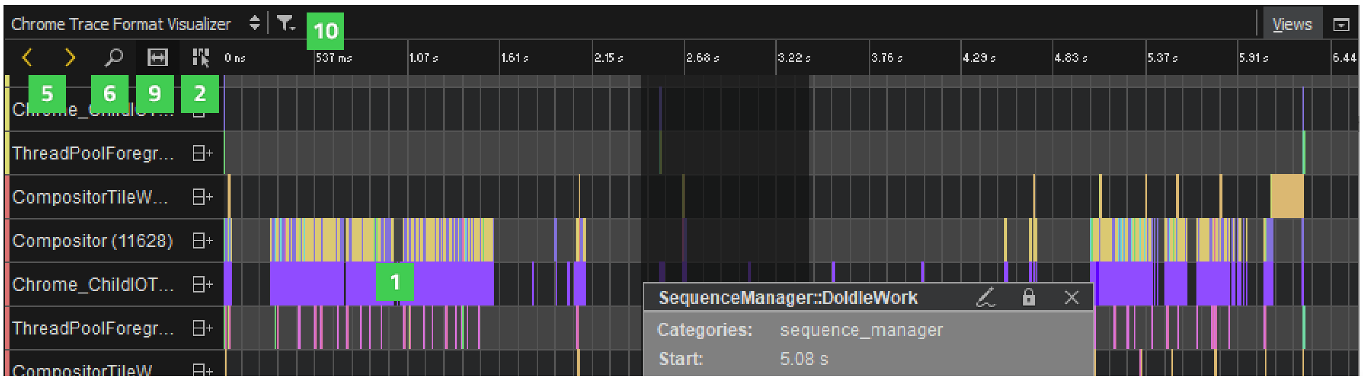
- 开始、结束、持续时间和即时事件
- 计数器事件（图形）
- 元数据事件（进程和线程名称）

打开文件

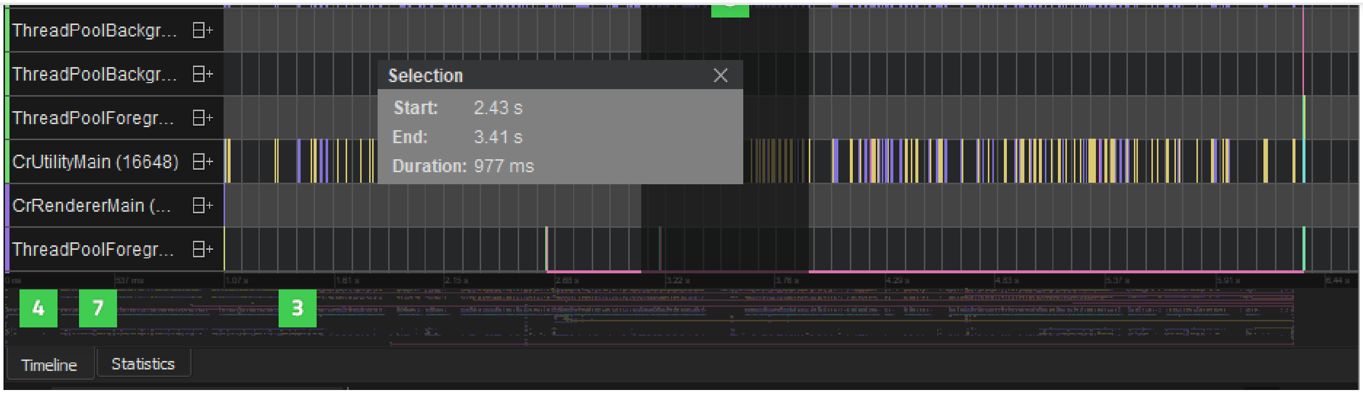
要打开 JSON 文件进行查看，请选择**分析**> Chrome **跟踪格式查看器**>**加载** JSON **文件**。

可视化事件

“**时间轴**”视图显示跟踪事件的图形表示形式和所有已记录事件的精简视图。



SequenceManager::DoIdleWork
Categories: sequence_manager
Start: 5.08 s



时间轴中的每个类别都描述应用程序中的一个线程。将光标移动到行上的事件 （1） 上以查看其持续时间和事件类别。若要仅在选择事件时显示信息，请禁用“鼠标**在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。若要在事件之间移动，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

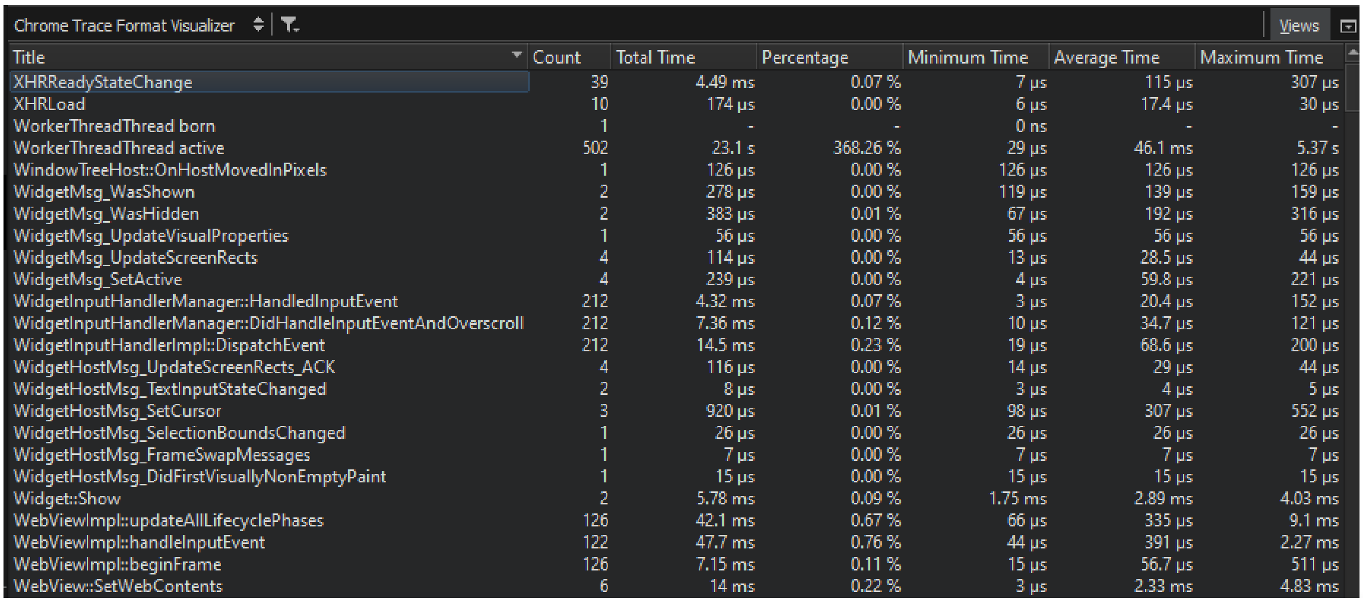
您可以选择事件范围 （8） 来查看它所表示的时间或放大跟踪的特定区域。选择“**选择范围**”按钮 （9） 以激活选择工具。然后在时间线中单击以指定事件范围的开头。拖动选择手柄以定义范围的末尾。

You can use event ranges also to measure delays between two subsequent events. Place a range between the end of the first event and the beginning of the second event. The **Duration** field displays the delay between the events in milliseconds.

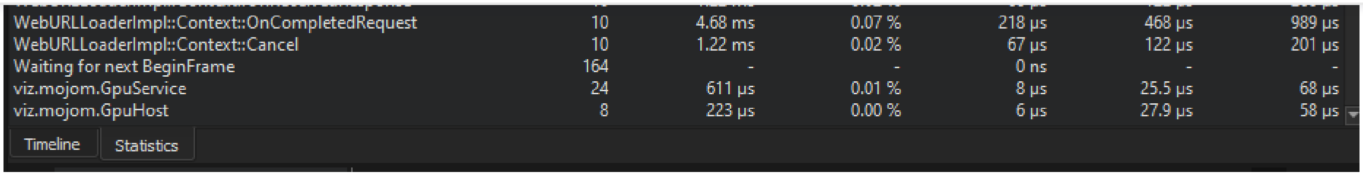
To zoom into an event range, double-click it.

To remove an event range, close the **Selection** dialog.

Viewing Statistics



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
Widget::Show	2	5.78 ms	0.09 %	1.75 ms	2.89 ms	4.03 ms
WebViewImpl::updateAllLifecyclePhases	126	42.1 ms	0.67 %	66 µs	335 µs	9.1 ms
WebViewImpl::handleInputEvent	122	47.7 ms	0.76 %	44 µs	391 µs	2.27 ms
WebViewImpl::beginFrame	126	7.15 ms	0.11 %	15 µs	56.7 µs	511 µs
WebView::SetWebContents	6	14 ms	0.22 %	3 µs	2.33 ms	4.83 ms



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

The **Statistics** view displays the number of samples each function in the timeline was contained in, in total and when on the top of the stack (called). This allows you to examine which functions you need to optimize. A high number of occurrences might indicate that a function is triggered unnecessarily or takes very long to execute.`self`

Collecting LTTng Data

LTTng is a tracing toolkit for Linux that you can apply on embedded Linux systems to find out how to optimize the startup time of an application.

Since Qt 5.13, Qt provides a set of kernel trace points and a tracing subsystem for custom user space trace points.

Configuring the Kernel

To use LTTng, you have to set the following configuration options for the kernel before building it:

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

We recommend that you set the following additional options:

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

In Yocto, you can activate the above options in **Menu** > **Config** > **Kernel Hacking** > **Tracers**.

Installing LTTng

After you build the kernel and deploy it on your device, you'll need to install the following LTTng packages on your device:

- `lttng-tools` to control the tracing session
- `lttng-modules` for kernel trace points
- `lttng-ust` for user space trace points

In Yocto, you just need to enable .`EXTRA_IMAGE_FEATURES += "tools profile"`

Building Qt with Tracepoints

Trace points are continuously being added to Qt versions. To use them, you need to build Qt yourself with the option.`configure -trace lttng` you specify the appropriate trace points as and call to enable them. Finally, you call to start tracing.`lttng create``lttng enable-channel kernel -k``kernel_events``lttng enable-event``lttng start`

You call to stop tracing. You can use to set the length of the session. After stopping, you can call to destroy the session.`lttng stop``sleep``lttng destroy`

You can write and run scripts that contain the above commands to start and stop full-stack tracing. You can use to execute the scripts.`systemd`

Enabling Trace Points

Data is recorded according to the trace points that you enable in the LTTng session. Usually, it is useful to enable scheduler switch, syscall, and Qt trace points.

Scheduler Switch Trace Points

Scheduler switch trace points are reached when an application is switched out due to predemption, for example, when another process gets the chance to run on the CPU core. Enable scheduler schwitch trace points to record the thread that is currently running and the process it belongs to, as well as the time when the process started and stopped.

Syscall Trace Points

Syscall trace points help you to understand why a scheduler switch happened. The following are examples of syscalls to trace:

- `openat` and map file descriptors to file names`close`
- `mmap` maps page faults to files
- `read` and are triggered by I/O operations`write`
- `nanosleep`, , and explain scheduler switches`futex``poll`
- `ioctl` controls the GPU and display

Converting LTTng Data to CTF

The ctf2ctf tool uses to parse binary Common Trace Format (CTF) and converts it to Chrome Trace Format (CTF). It performs the following custom tasks to make the recording more human-readable:`babeltrace`

- Map file descriptors to file names
- Map page faults to file names
- Annotate interrupts and block devices with names
- Convert UTF-16 QString data to UTF-8 strings
- Count memory page allocations

To generate JSON files that contain the trace data in Chrome Trace Format, enter the following command on the command line:

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

Qt DOCUMENTATION

Qt The Qt Company

Contact Us

Company

About Us
Investors
Newsroom
Careers
Office Locations

Licensing

Terms & Conditions
Open Source
FAQ

Support

Support Services
Professional Services
Partners
Training

For Customers

Support Center
Downloads
Qt Login
Contact Us
Customer Success

Community

Contribute to Qt
Forum
Wiki
Downloads
Marketplace

Feedback Sign In