

《dynaTrace Ajax 教程 – 进阶篇》

大灰狼系列分享



2010-4-19

作者:大灰狼堡森

《dynaTrace Ajax 教程 – 进阶篇》

大灰狼系列分享

PRE

在 dynaTrace Ajax 使用指南-基础篇中，我们对 dynaTrace 主要的功能进了一番介绍，在进阶篇中，我们将更深层的去了解如何使用 dynaTrace 来分析更底层的问题，这将会对 JavaScript 开发、调试和性能测试大有帮助。



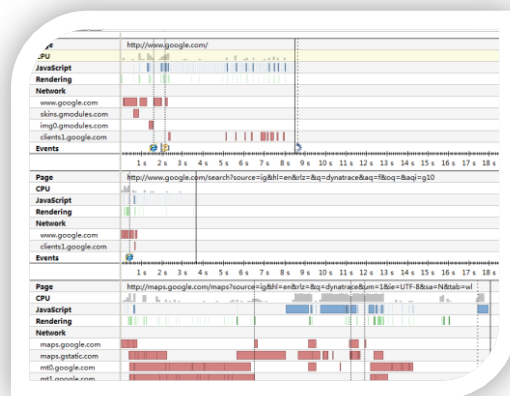
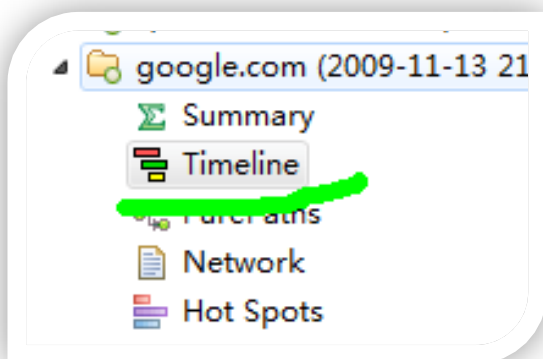
说明

因环境差异每次追踪文件都会不同，为了保证本教程内所有内容一致行，请将本教程附带的 step_by_step_google.dtas 文件通过 Import Session 导入，如下图：

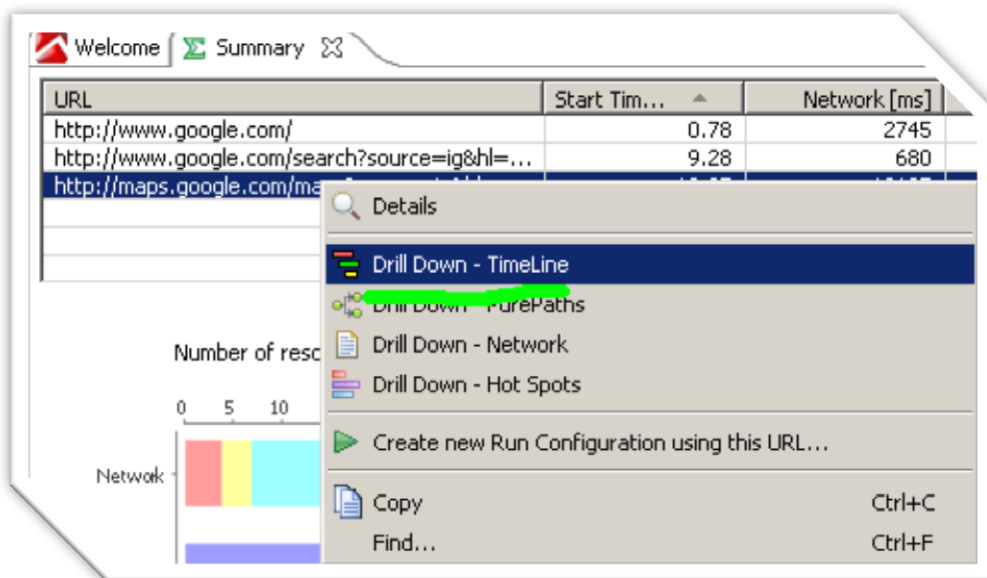


TimeLine 视图 – 进一步查看页面生命周期事件

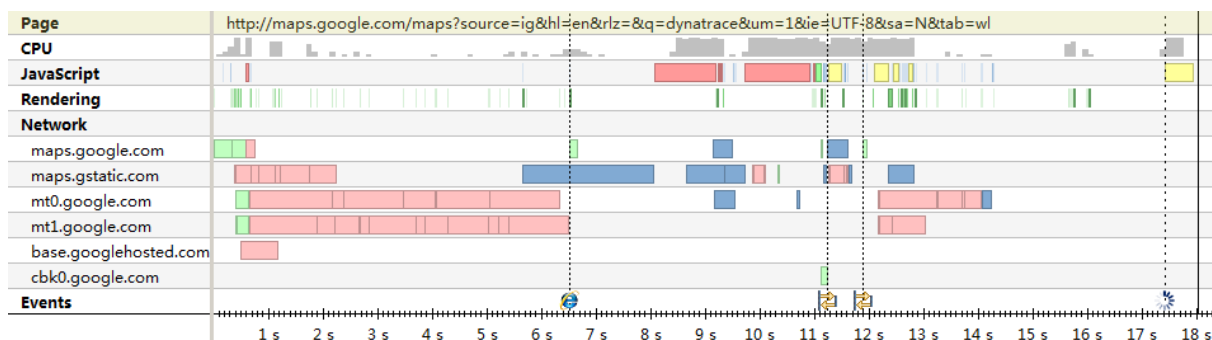
打开 TimeLine 视图的方式很简单，我们可以通过双击左边树形菜单的 TimeLine 节点来打开包含整个 session 所有信息的 TimeLine 视图，如下图。



如果你只想查看某个 URL 的 TimeLine，右键点击该 URL 后选择 Drill-Down TimeLine，如下图。



这将会打开针对于某个 URL 请求的 TimeLine 视图，并且会自动分开显示不同域（domain）的网络请求。在工具栏和上下文菜单中，你可以开启一些可选选项，比如内容类型着色、JavaScript 触发器或显示附加的事件（如鼠标移动、鼠标点击、键盘按键等）。在下图中就是一个带有附加选项的 TimeLine 视图：



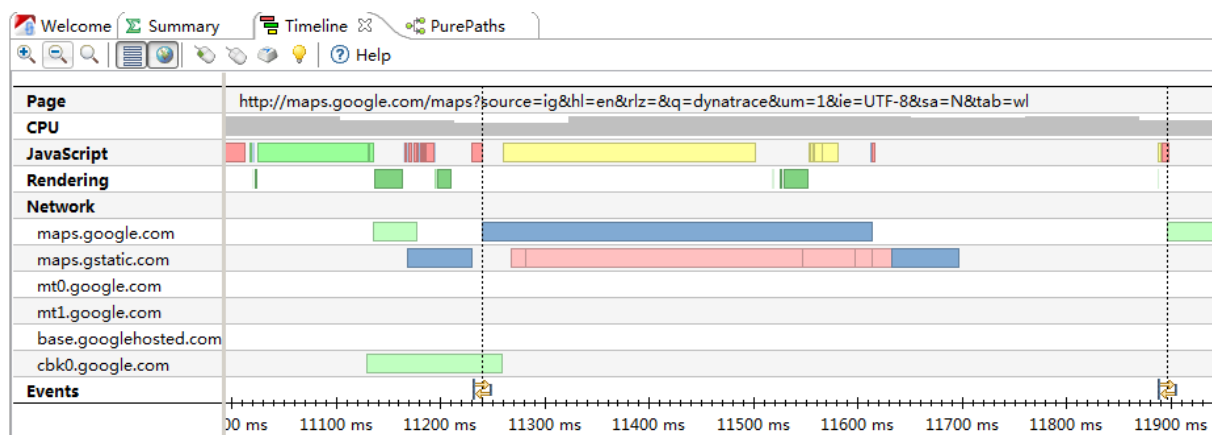
在这个视图中，我们可以了解到：

- 网络请求是并行从 6 个不同的域中下载的。
- 在 onLoad 事件触发前使用了 6.5s(通过小 IE 标志能看到)
- 从 maps.gstatic.com 下载的主文件 main.js 用了 2.42s（鼠标停留在上面会有更详细的资料）
- 紧接着 main.js，我们可以看到，脚本实际运行时间是 1.1s，期间有两个下载（1s，红色线 JavaScript 行）和 2 个附加的 JavaScript 执行（2s）
- CPU 行显示了运行期间 CPU 的使用情况
- 事件行显示了我们的鼠标点击事件、XmlHttpRequest 事件和 onUnload 事件有关信息。



首先让我们放大从第一次鼠标点击到 XmlHttpRequest 请求之间的 TimeLine。在这个例子中就是从 11s 到 12s 之间的部分（从 JavaScript 行绿色的那条开始，之后在 Network 行相对较长的蓝色线表示 XmlHttpRequest 请求，鼠标悬停上去有详细信息哈）。这里教大家一个小技巧，在实现线

上如果想放大某一段，用鼠标在起始时间按住，然后托到终点时间，之间的部分就会被放大。从又往左托会缩小。如下图所示。（用鼠标滑轮也可以放大缩小）

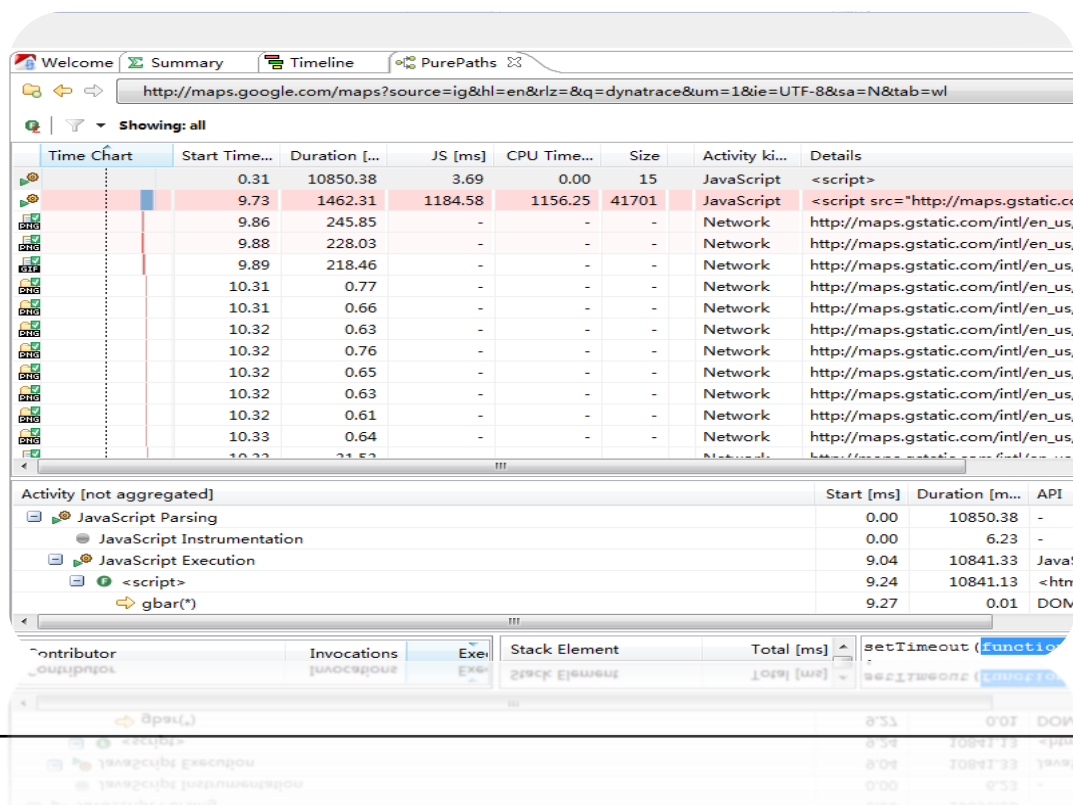


TimeLine 现在显示了鼠标点击事件、一个 XMLHttpRequest 事件，紧接着一个 onEvent 和另一个 XMLHttpRequest(XHR)。用鼠标悬停在事件上将会显示哪些 DOM 元素触发了什么事件。在 JavaScript 行上悬停将会看到执行事件句柄（Event Handlers）使用了多长时间，在 Network 行上悬停显示哪些附加资源被下载。我们还能看到浏览器进行了哪些类型的呈现（Rendering）。在本例中，我们可以了解到：第一次鼠标点击引起了附加的内容被下载 --- 包括另一个 JavaScript 文件（来自 maps.gstatic.com）。当这个 JavaScript 文件下载后就立刻执行，并且触发到一个 XHR 请求。我们也可以知道 onEvent 时间句柄被触发和运行的时间为 240ms。

PurePath 视图 - 解决 JavaScript、DOM 和 Ajax 的问题




在 TimeLine 视图(Timeline View)中我们可以更深一层的了解浏览器和网络的活动，了解哪些 JavaScript 被真正的执行，谁触发了 XHR 请求。如果你右键点击 TimeLine，你可以选择“Drill Down To TimeFrame”，这将会进入到 PurePath 视图来显示当前被放大的 TimeLine 片段内的所有活动情况，如下图所示：

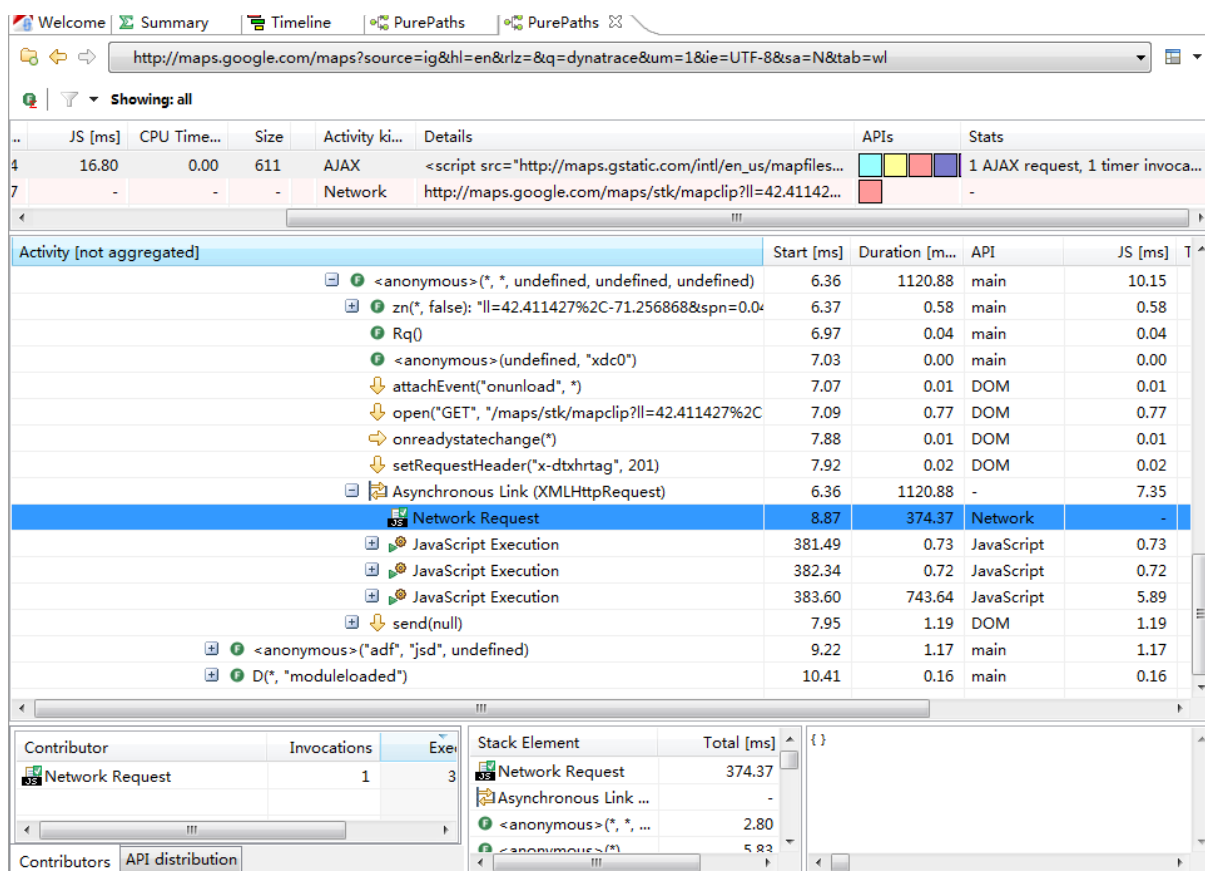




在上面的视图中我们可以看到选择的 TimeLine 片段内所有的浏览器的活动，包括 Script 标签或事件句柄触发了哪些 JavaScript 的执行，还有网络请求和呈现的时间。当一些定时器（timer）或 Ajax 请求被触发时，对应的行会被高亮显示，最右边的 Stats 列显示了相关的信息。其它的大家可以自己尝试，内容很丰富很强大。


在 PurePath 列表选择一个活动将会更新下面的相关视图。PurePath 树显示了 JavaScript 真实的运行路径，包括不同方法的执行时间、参数、返回值。其中不仅有 JavaScript 方法，还包括进入 DOM 和通过 XHR 调用 Ajax 请求的详细信息。

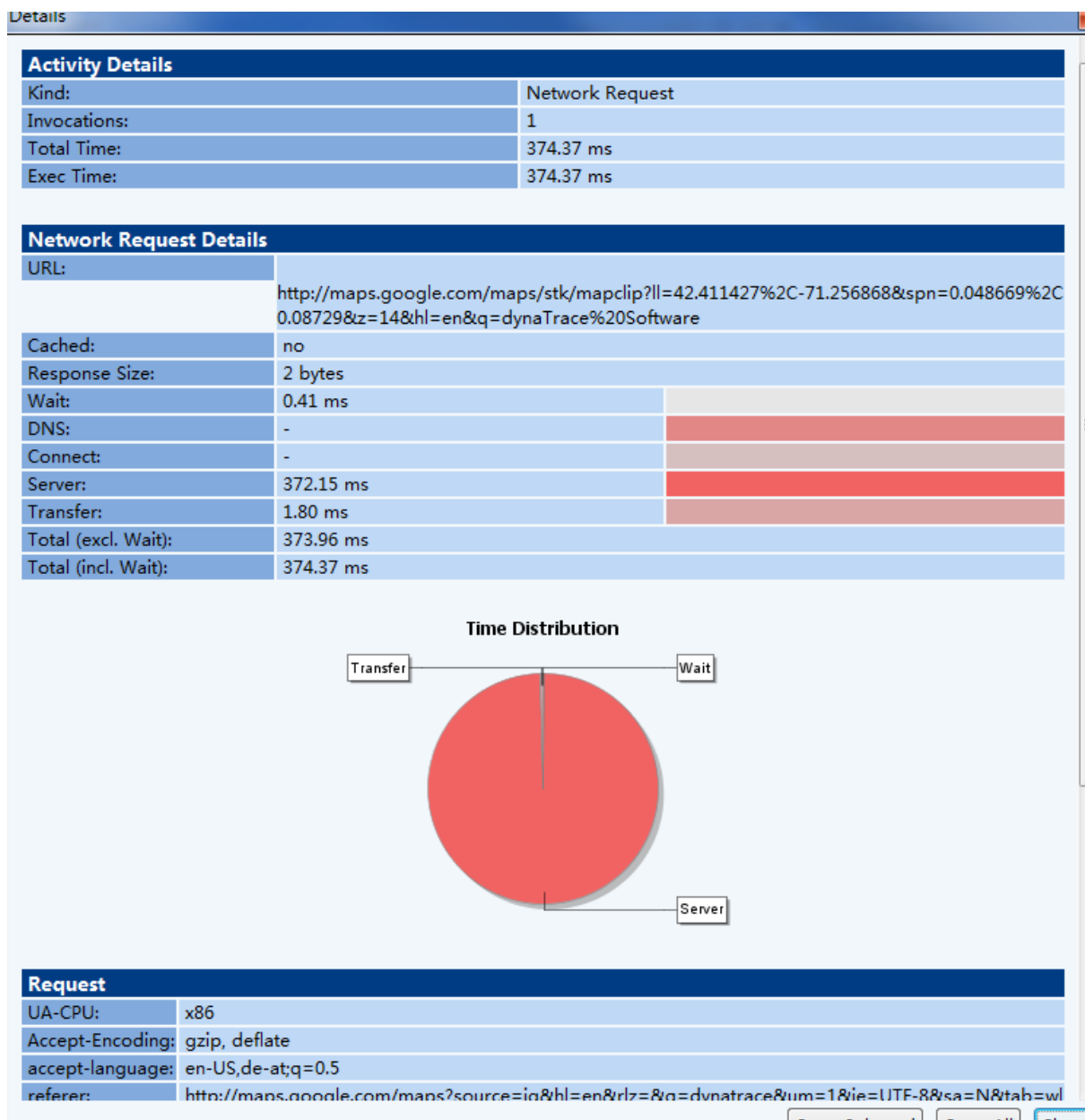
让我们退回一步到之前说到的 TimeLine 视图。我现在对这个发送的 Ajax 请求感兴趣。双击左边的  来进入它的 PurePath，即追踪到的执行这个 XHR 的 JavaScript 详情，如下图：



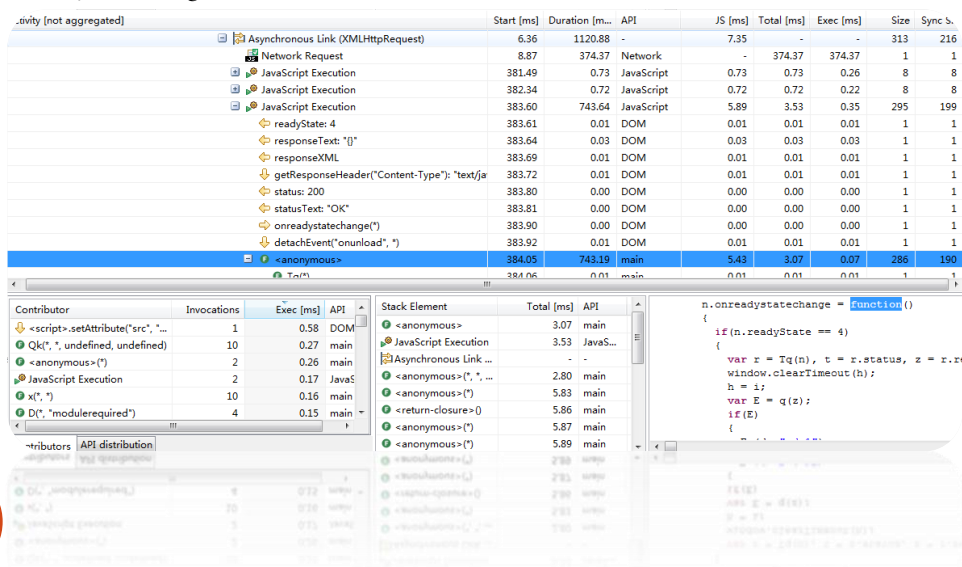
在上面我们能看见这些代码所属的 JavaScript 文件。在树视图中我们能看见所有的 JavaScript 追踪，其中就有我们要找的触发 XHR 请求的方法，包括参数。Duration 列告诉我们 JavaScript 总



共运行了 1127ms（最上面的一条，树根处），包括等待 XHR 请求返回和等待 JavaScript 定时器(timer)的时间。右键 Network Request 节点（就是那个  Network Request）选 Details 打开细节视图，将会显示 Http 请求和响应的头、请求链接、等待、服务器处理和数据传输的精确时间，也包括从服务器返回的真实内容。如下图所示：



比较有趣的是，在这个请求使用了 372ms 响应一个空 JSON 对象。从这里我们可以继续分析 AJAX 响应究竟做了些什么。在 PurePath 树中我们找到图中所示的节点，可以看到 onreadystatechange 句柄。下图显示了树中的这个句柄和右下角的真正的源代码。



我们可能对左下角的 contributor 列表比较感兴趣，它显示了所有选择的子树（或节点）中所有的 JavaScript 活动。双击 contributor 列表中的某一项，会自动定位到对应的树节点。，让我们双击 contributor 列表中的 setTimeout 节点：

Activity [not aggregated]	Start [ms]	Duration [m...	API
	385.24	742.00	DOM
	385.26	741.98	JavaScript
	385.26	741.98	-
	1124.88	2.36	JavaScript
	1124.89	2.34	main
	1124.90	2.30	main
	1124.91	0.73	main
	1124.91	0.00	main
	1124.94	0.60	main
	1124.95	0.00	main
	1124.96	0.22	main
	1125.19	0.34	main

Contributor	Invocations	Exec [ms]	API
setTimeout(*, 0): 56639482	1	0.02	DOM
<anonymous>	1	0.02	main
C(*): 7	7	0.02	main
<anonymous>(<script>, "er...	1	0.02	main
lo(<script>, "error", false)	1	0.02	main
<anonymous>(*)	1	0.02	main
<return-closure>()	1	0.02	main
C(*): 1	5	0.02	main
U(<script>, "error", *, *)	1	0.02	main

Stack Element	Total [ms]	API
↓ setTimeout(*, 0): 56...	0.17	DOM
nn(*, *, 0, undefined...	0.21	main
<anonymous>("htt...	0.32	main
<anonymous>(*, *)	0.36	main
j(*, *)	0.38	main
<anonymous>(*)	0.41	main
<anonymous>(*)	0.42	main
<anonymous>("mp...	0.44	main
<anonymous>(*)	0.58	main
<return-closure>()	0.60	main

在这里我们能观察到：



- 一个定时器(Timer)用了 740ms 触发到一个 timer 句柄。
- 一个标签被动态的创建，并添加到 head DOM 元素中，用以告诉浏览器在执行结束后下载这个脚本。

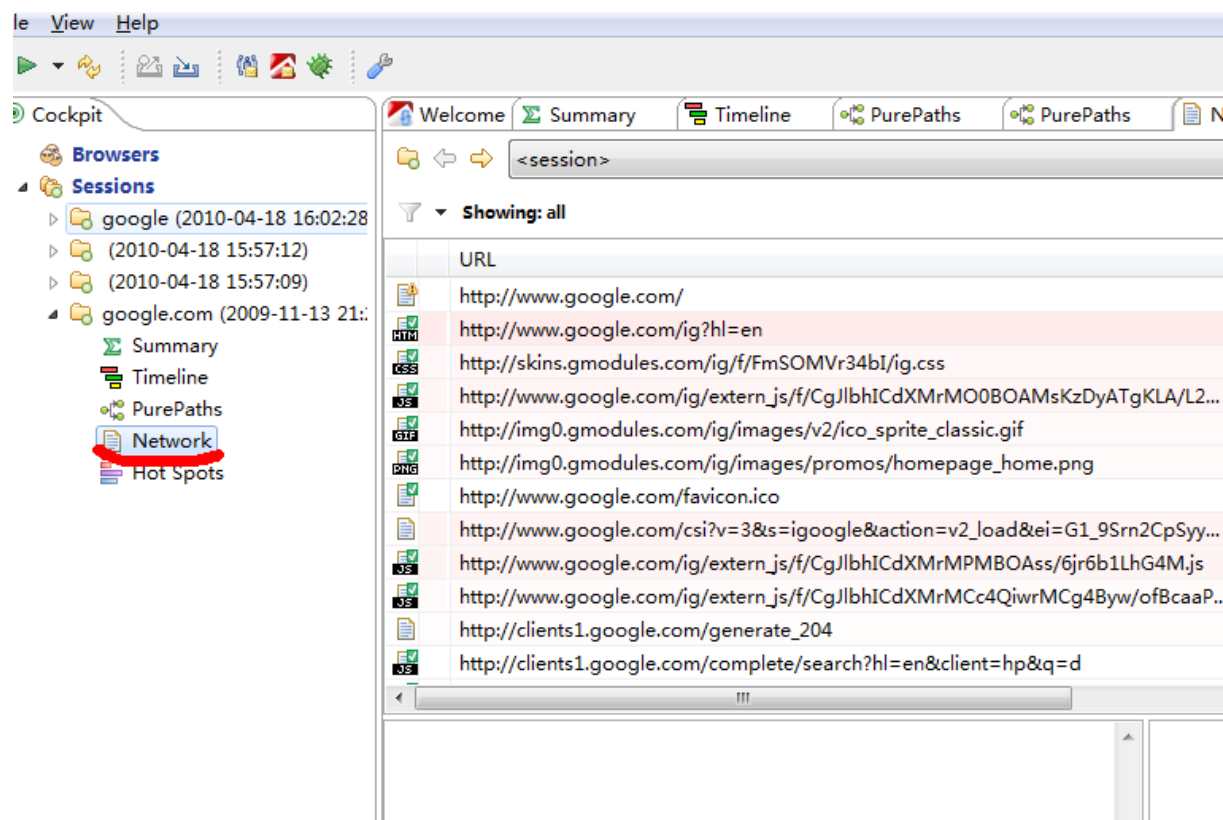
PurePath 视图提供了不同的方式来分析数据。你可以通过直接用键盘输入关键字来过滤和搜索每张表或树中想要的内容。

Find: Next Previous Filter

Contributor API distribution

Network 视图

Network 视图显示了在整个浏览器 session 内或不同页面内的所有网络请求。在左边树形主菜单中通过双击 Network 节点可以打开 Network 视图。



我们能看到不同项目的颜色不同，越红越暴力的表示请求时间越长。默认情况下会以 TimeLine 上的发生顺序来排列，你可以点击任何一列来进行排序。比如你看，我可以酱紫把最红最暴力的都放到前面，如下图所示。

	Cach...	Kind	Status	MIME	Size [byt...	Time Chart	Start T...	Total [s]	Netwo...	W
058&z=14&s=G	no	Net	200 ...	image/png	18918		13.60	5.76	1.18	
055&z=14&s=Gal	no	Net	200 ...	image/png	12509		13.59	1.51	1.49	
056&z=14&s=Galile	no	Net	200 ...	image/png	9613		25.13	1.05	1.05	
057&z=14&s=	no	Net	200 ...	image/png	17962		13.60	4.37	1.16	
054&z=14&s=Ga	no	Net	200 ...	image/png	13604		25.12	1.08	1.08	
058&z=14&s=Gali	no	Net	200 ...	image/png	20328		13.60	5.67	1.28	
055&z=14&s=G	no	Net	200 ...	image/png	13114		13.59	1.73	1.71	
055&z=14&s=Galile	no	Net	200 ...	image/png	15075		13.59	1.58	1.56	
058&z=14&s=Galile	no	Net	200 ...	image/png	15122		13.60	5.07	1.64	
s2/%7Bmod_jslinker,...	no	Net	200 ...	text/javascript	180665		21.61	1.07	1.06	
056&z=14&s=Gali	no	Net	200 ...	image/png	12908		13.60	3.43	1.92	

对于不同的请求（列表中不同的条目）我们可以看到

- 它是否来自浏览器缓存（通过 Cached 列）
- 请求的类型（Network 或 Ajax）
- HTTP 状态
- MIME 类型



- 大小等等等等（不一一列了，太多了）

对于每一个条目，鼠标右键可以点 PurePath 进入到 PurePath 视图。

Hotspot 视图 – 最慢最影响性能的地方在哪里？

最后一个有趣的视图是 Hotspot 视图。通过老地方打开（左边树形做菜单，找不到的话估计你也看不到这儿了），也可以通过 Summary 视图中某个 URL 的右键菜单中进入到某个 URL 的 Hotspot 视图。我们前者打开来看看所有我访问过的页面的 JavaScript、DOM、Render 活动：

Contributor	API	Invocations	Exec [%]	Exec [ms]	Exec Avg [ms]	Total Sum [ms]	Total Avg [ms]
Rendering (Drawing)	-	103		868.70	8.43	868.70	8.43
Rendering (Calculating flow layout)	-	946		162.44	0.17	165.06	0.17
Rendering (Calculating generic lay...	-	179		159.77	0.89	162.26	0.90
JavaScript Parsing	-	45		139.26	3.09	2965.11	65.89
<anonymous>(<body>)	main	1		74.54	74.54	259.90	259.90
<anonymous>(<div>)	main	64		67.89	1.06	302.61	4.72
<anonymous>(*)	main	127		61.39	0.48	83.31	0.65
<anonymous>(<div>)	main	1293		59.01	0.04	181.27	0.14
<return-closure>()	main	658		58.66	0.08	201.39	0.30

Back Traces	API	Invocations	Exec [%]	Exec [ms]	Exec Avg [ms]	# Caller Invoca...
Rendering (Calculating flow layout #5)	-	946		162.44	0.17	0
Rendering (Rendering activities)	-	464		111.36	0.24	43
<input>.offsetTop: 20	DOM	11		16.77	1.52	10
<div>.offsetLeft: 636	DOM	28		10.12	0.36	2
<return-closure>()	main	28		10.12	0.36	2
<anonymous>(*)	main	28		10.12	0.36	2
<div>.offsetWidth: 1017	DOM	45		3.43	0.07	3
.getAttribute("height"): 1	DOM	6		2.06	0.47	1
<div>.offsetLeft: 0	DOM	82		2.49	0.03	2
<div>.offsetWidth: 89	DOM	14		2.17	0.15	3
<div>.offsetLeft: -205	DOM	40		1.89	0.04	3
<html>.clientHeight: 887	DOM	49		1.84	0.03	2
<div>.offsetTop: 0	DOM	17		1.49	0.08	1

Forward Traces	API	Invocations	Sync Size	Size	Total [ms]	Exec [ms]	JS [ms]
Rendering (Calculating flow layout)	-	946	1781	1781	165.06	162.44	165.06
Rendering (Scheduling layout task)	-	835	835	835	2.61	2.61	2.61


```

return function ()
{
    return b.apply(a || this, arguments);
};
return;
}
function bm(a,b)

```

上面的表格中统计了所有 JavaScript、DOM 和 Rendering 的活动（Activity）。我们能看到一共有 103 个会话活动、946 个 Reflow 活动和 1293 个 div 标签上的匿名函数调用。双击上面的表格中的某一项，会看到相应的 Back Trace 和 Forward Trace，其中 Back Trace 显示了谁引发了所选的活动，而 Forward Trace 显示了这个活动触发了哪些其它的活动。点击 Back Trace 或 Forward Trace 中树的节点，会在最下面显示相关的 JavaScript 源代码。

Bosn 的总结

本文对 dynaTrace Ajax 的功能进行了更深入的讲解，包括所有的视图介绍（Purepath 视图、Network 视图、Hotspot 视图等）。熟练的使用这些工具将会对开发和性能优化有所帮助。希望本文能帮助到大家，谢谢。



如果文中有误，欢迎批评指正。

Hi: bosnma

E-mail: bosnma@live.cn

感谢您阅读大灰狼系列分享文档。



参考文献：<http://blog.dynatrace.com/2009/11/17/a-step-by-step-guide-to-dynatrace-ajax-edition-available-today-for-public-download/>



THE END