你好，我是李兵。

作为一名前端工程师，除了需要编写功能性的代码以外，我们还需要关注Web应用的性能问题，我们应该有能力让我们的Web应用占用最小的资源，并以最高性能运行，这也是前端工程师进阶的必要能力。既然性能这么重要，那么我们今天要来聊聊Web性能问题。

到底什么是Web性能？

我们看下wiki对Web 性能的定义：

> Web 性能描述了Web应用在浏览器上的加载和显示的速度。

因此，当我们讨论Web性能时，其实就是讨论Web应用速度，关于Web应用的速度，我们需要从两个阶段来考虑：

- 页面加载阶段；
- 页面交互阶段。

在本文中，我们会将焦点放到第一个阶段：页面加载阶段的性能，在下篇文章中，我们会来重点分析页面交互阶段的性能。

性能检测工具：Performance vs Audits

要想优化Web的性能，我们就需要有监控Web应用的性能数据，那怎么监控呢？

如果没有工具来模拟各种不同的场景并统计各种性能指标，那么定位Web应用的性能瓶颈将是一件非常困难的任务。幸好，Chrome为我们提供了非常完善的性能检测工具：**Performance**和**Audits**，它们能够准确统计页面在加载阶段和运行阶段的一些核心数据，诸如任务执行记录、首屏展示花费的时长等，有了这些数据我们就能很容易定位到Web应用的**性能瓶颈** 。

首先Performance非常强大，因为它为我们提供了非常多的运行时数据，利用这些数据我们就可以分析出来Web应用的瓶颈。但是要完全学会其使用方式却是非常有难度的，其难点在于这些数据涉及到了特别多的概念，而这些概念又和浏览器的系统架构、消息循环机制、渲染流水线等知识紧密联系在了一起。

相反，Audtis就简单了许多，它将检测到的细节数据隐藏在背后，只提供给我们一些直观的性能数据，同时，还会给我们提供一些优化建议。

Perfomance能让我们看到更多细节数据，但是更加复杂，Audits就比较智能，但是隐藏了更多细节。为了能够让你循序渐进地理解内容，所以本节我们先从简单的Audits入手，看看如何利用它来检测和优化页面在加载阶段的性能，然后在下一节我们再来分析Perfomance。

检测之前准备工作

不过在检测Web的性能指标之前，我们还要配置好工作环境，具体地讲，你需要准备以下内容：

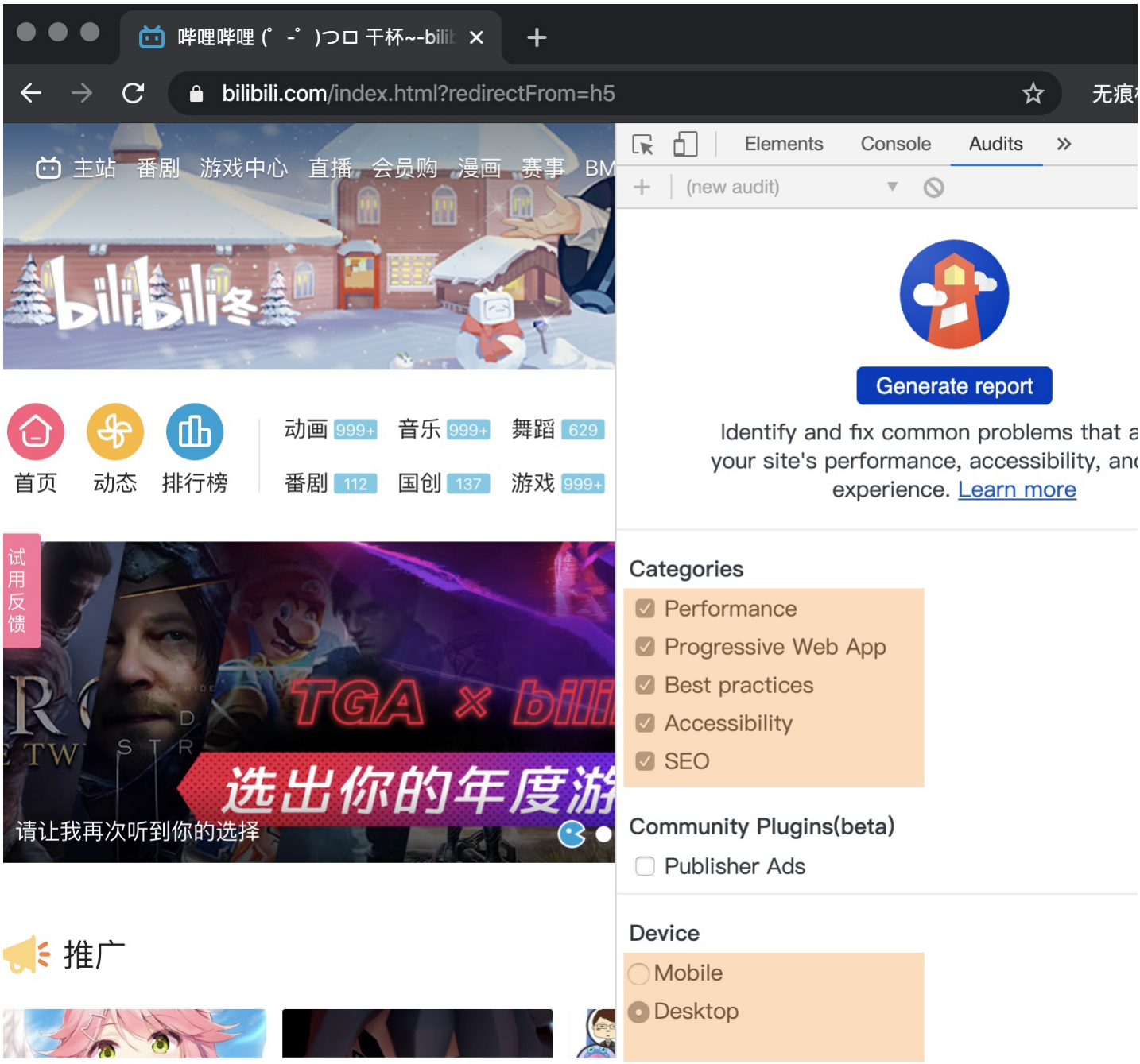
- 首先准备Chrome Canary版的浏览器，Chrome Canary是采用最新技术构建的，它的开发者工具和浏览器特性都是最新的，所以我推荐你使用Chrome Canary来做性能分析。当然你也可以使用稳定版的Chrome。
- 然后我们需要在Chrome的隐身模式下工作，这样可以确保我们安装的扩展、浏览器缓存、Cookie等数据不会影响到检测结果。

利用Audits生成Web性能报告

环境准备好了之后，我们就可以生成站点在加载阶段的性能报告了，这里我们可以拿B站作为分析的列子。

- 首先我们打开浏览器的隐身窗口，Windows系统下面的快捷键是Control+Shift+N，Mac系统下面的快捷键是Command+Shift+N。
- 然后在隐身窗口中输入B站的网站。
- 打开Chrome的开发者工具，选择Audits标签。

最终打开的页面如下图所示：



Audits界面

观察上图中的Audits界面，我们可以看到，在生成报告之前，我们需要先配置Audits，配置模块主要有两部分组成，一个是**监测类型(Categories)**，另外一个是**设备类型(Device)**。

监控类型(Categories)是指需要监控哪些内容，这里有五个对应的选项，它们的功能分别是：

- 监测并分析Web性能(**Performance**)；
- 监测并分析PWA(**Progressive Web App**)程序的性能；
- 监测并分析Web应用是否采用了最佳实践策略(**Best practices**)；
- 监测并分析是否实施了无障碍功能(**Accessibility**)，无障碍功能让一些身体有障碍的人可以方便地浏览你的Web应用。
- 监测并分析Web应用是否采实施了SEO搜索引擎优化(**SEO**)。

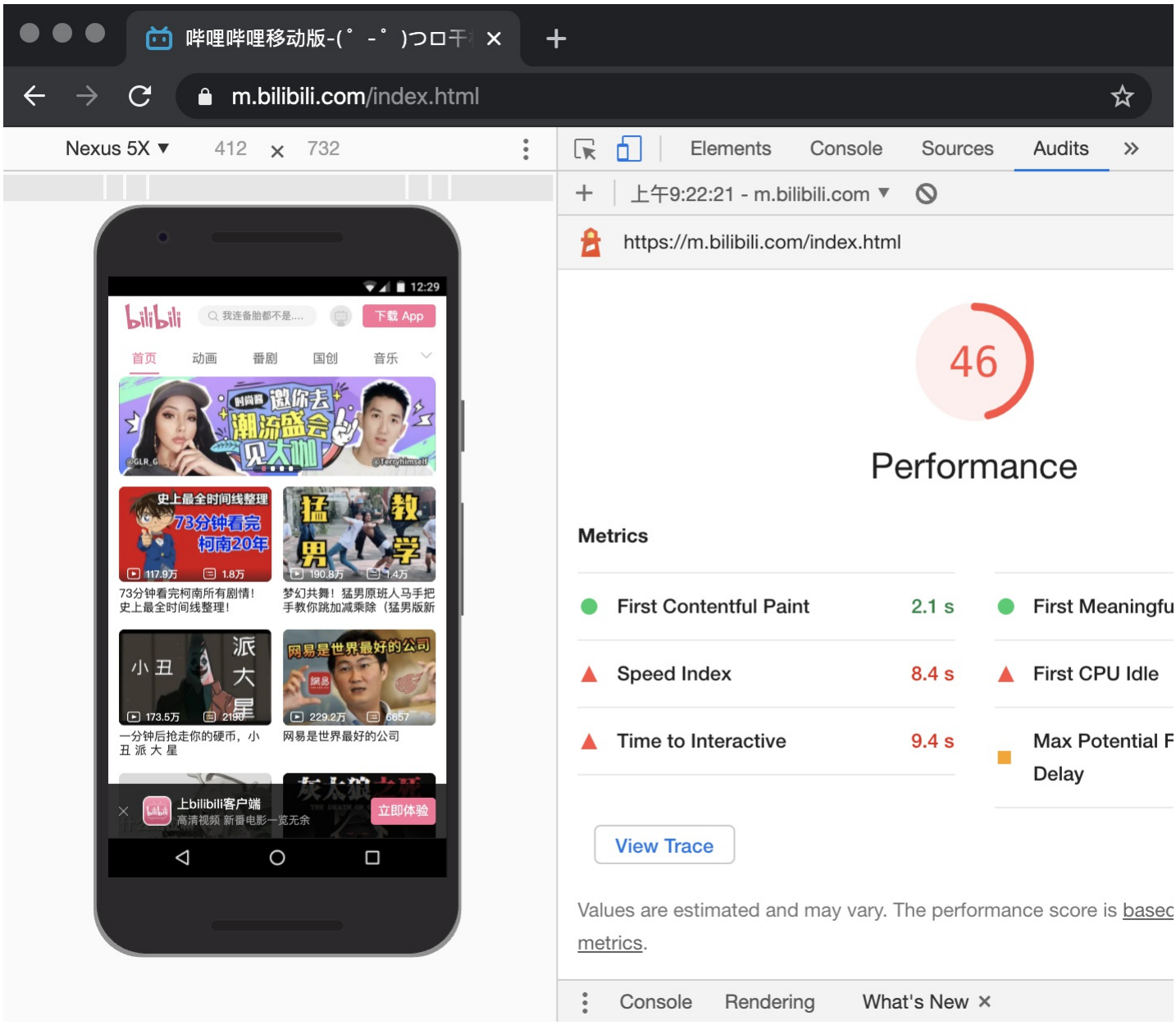
本文我们只需要关注Web应用的加载性能，所以勾选第一个Performance选项就可以了。

再看看**设备(Device)部分**，它给了我们两个选项，Moblie选项是用来模拟移动设备环境的，另外一个Desktop选项是用来模拟桌面环境的。这里我们选择移动设备选项，因为目前大多数流量都是由移动设备产生的，所以移动设备上的Web性能显得更加重要。

配置好选项之后，我们就可以点击最上面的生成报告(Generate report)按钮来生成报告了。

解读性能报告

点击生成报告的按钮之后，我们大约需要等待一分钟左右，Audits就可以生成最终的分析报告了，如下图所示：



生成的报告图

观察上图的分析报告，中间圆圈中的数字表示该站点在加载过程中的总体Web性能得分，总分是100分。我们目前的得分为46分，这表示该站点加载阶段的性能还有很大的提升空间。

Audits除了生成性能指标以外，还会分析该站点并提供了很多优化建议，我们可以根据这些建议来改进Web应用以获得更高的得分，进而获得更好的用户体验效果。

既能分析Web性能得分又能给出优化建议，所以Audits的分析报告还是非常有价值的，那么接下来，我们就来解读下Audits生成的性能报告。

报告的第一个部分是**性能指标(Metrics)**，如下图所示：

Metrics

First Contentful Paint	2.1 s	First Meaningful Paint	2.1 s
Speed Index	8.4 s	First CPU Idle	7.2 s
Time to Interactive	9.4 s	Max Potential First Input Delay	230 ms

View Trace

Values are estimated and may vary. The performance score is based only on these metrics.

性能指标

观察上图，我们可以发现性能指标下面一共有六项内容，这六项内容分别对应了从Web应用的加载到页面展示完成的这段时间中，各个阶段所消耗的时长。在中间还有一个View Trace按钮，点击该按钮可以跳转到Performance标签，并且查看这些阶段在Performance中所对应的位置。最下方是加载过程中各个时间段的屏幕截图。

报告的第二个部分是**可优化项(Opportunities)**，如下图所示：

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Eliminate render-blocking resources	0.76 s
Preconnect to required origins	0.38 s
Warnings: A preconnect <link> was found for "https://api.bilibili.com" but was not used by the browser. Check that you are using the `crossorigin` attribute properly.	
Properly size images	0.16 s

可优化项(Opportunities)

这些可优化项是Audits发现页面中的一些可以直接优化的部分，你可以对照Audits给的这些提示来优化你的Web应用。

报告的第三部分是**手动诊断(Diagnostics)**，如下图所示：

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- Avoid enormous network payloads — Total size was 4,257 KB
- Serve static assets with an efficient cache policy — 7 resources found
- Avoid an excessive DOM size — 892 elements
- Avoid chaining critical requests — 12 chains found
- Keep request counts low and transfer sizes small — 236 requests • 4,257 KB

Passed audits (16)

手动诊断(Diagnostics)

在手动诊断部分，采集了一些可能存在性能问题的指标，这些指标可能会影响到页面的加载性能，Audits把详情列出来，并让你依据实际情况，来手动排查每一项。

报告的最后一部分是**运行时设置(Runtime Settings)**，如下图所示：

Runtime Settings

URL	https://m.bilibili.com/index.html
Fetch time	Nov 30, 2019, 9:22 AM GMT+8
Device	Emulated Nexus 5X
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3980.0 Safari/537.36
User agent (network)	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Mobile Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1612

Generated by **Lighthouse** 5.7.0 | File an issue

运行时设置(Runtime Settings)

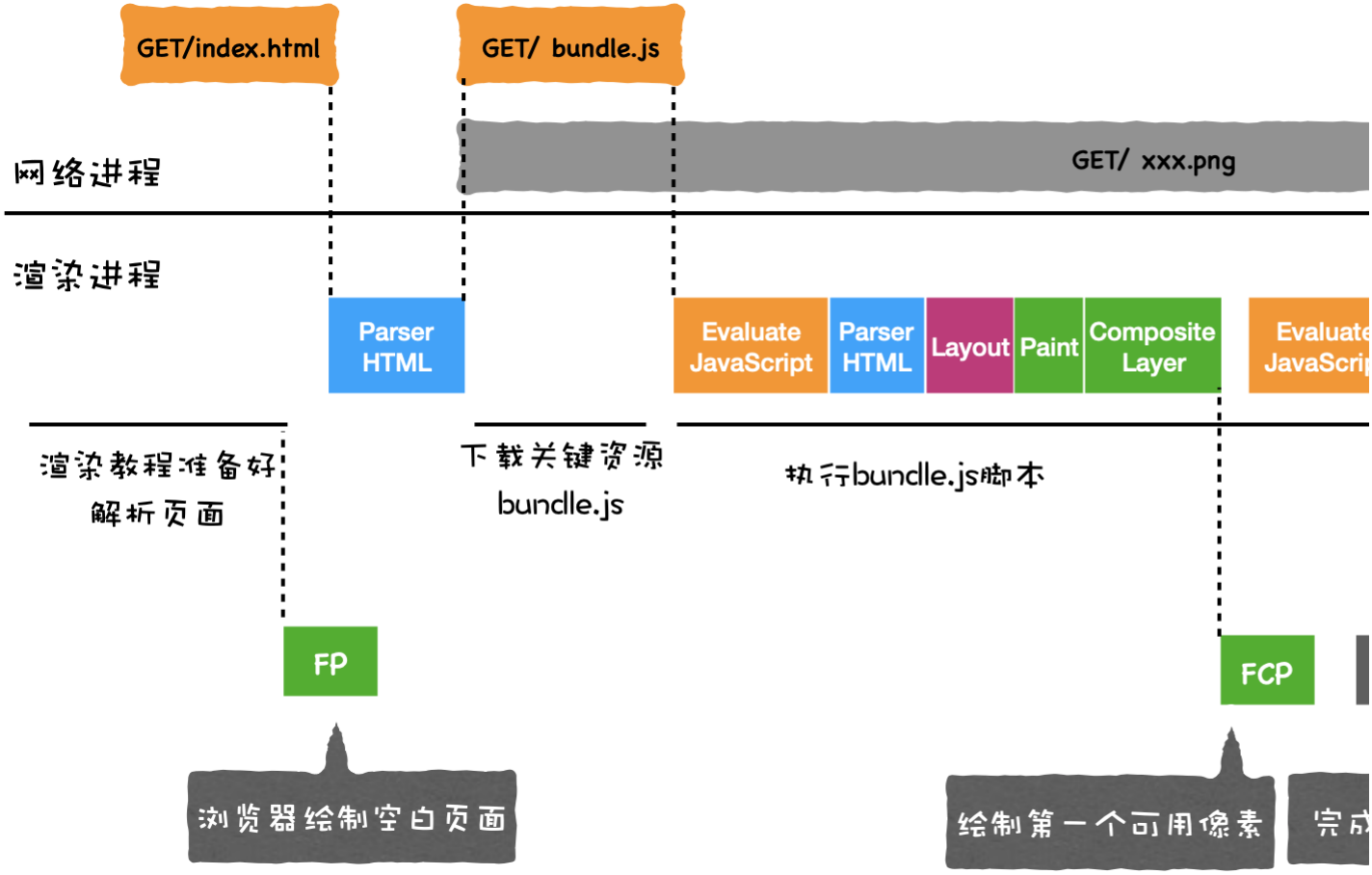
观察上图，这是运行时的一些基本数据，如果选择移动设备模式，你可以看到发送网络请求时的User Agent 会变成设备相关信息，还有会模拟设备的网速，这个体现在网络限速上。

根据性能报告优化Web性能

现在有了性能报告，接下来我们就可以依据报告来分析如何优化Web应用了。最直接的方式是想办法提高性能指标的分数，而性能指标的分数是由六项指标决定的，它们分别是：

1. 首次绘制(First Paint)；
2. 首次有效绘制(First Meaningfull Paint)；
3. 首屏时间(Speed Index)；
4. 首次CPU空闲时间(First CPU Idle)；
5. 完全可交互时间(Time to Interactive)；
6. 最大估计输入延时(Max Potential First Input Delay)。

那么接下来我会逐一分析六项指标的含义，并讨论如何提升这六项指标的数值。这六项都是页面在加载过程中的性能指标，所以要弄明白这六项指标的具体含义，我们还得结合页面的加载过程来分析。一图胜过千言，我们还是先看下面这张页面从加载到展示的过程图：



页面加载过程

观察上图的页面加载过程，我们发现，在渲染进程确认要渲染当前的请求后，渲染进程会创建一个空白页面，我们把创建空白页面的这个时间点称为**First Paint**，简称**FP**。

然后渲染进程继续请求关键资源，我们在《25丨页面性能：如何系统地优化页面？》这节中介绍过了关键资源，并且知道了关键资源包括了JavaScript文件和CSS文件，因为关键资源会阻塞页面的渲染，所以我们需要等待关键资源加载完成后，才能执行进一步的页面绘制。

上图中，bundle.js是关键资源，因此需要完成加载之后，渲染进程才能执行该脚本，然后脚本会修改DOM，引发重绘和重排等一系列操作，当页面中绘制了第一个像素时，我们把这个时间点称为**First Content Paint**，简称**FCP**。

接下来继续执行JavaScript脚本，当首屏内容完全绘制完成时，我们把这个时间点称为**Largest Content Paint**，简称**LCP**。

在FCP和LCP中间，还有一个FMP，这个是首次有效绘制，由于FMP计算复杂，而且容易出错，现在不推荐使用该指标，所以这里我们也不做过多介绍了。

接下来JavaScript脚本执行结束，渲染进程判断该页面的DOM生成完毕，于是触发DOMContentLoad事件。等所有资源都加载结束之后，再触发onload事件。

好了，以上就是页面在加载过程中各个重要的时间节点，了解了这些时间节点，我们就可以来聊聊性能报告的六项指标的含义并讨论如何优化这些指标。

我们先来分析下**第一项指标FP**，如果FP时间过久，那么直接说明了一个问题，那就是页面的HTML文件可能由于网络原因导致加载时间过久，这块具体的分析过程你可以参考《21丨Chrome开发者工具：利用网络面板做性能分析》这节内容。

第二项是FMP，上面也提到过由于FMP计算复杂，所以现在不建议使用该指标了，另外由于LCP的计算规则简单，所以推荐使用LCP指标，具体文章你可以参考这里。不过是FMP还是LCP，优化它们的方式都是类似的，你可以结合上图，如果FMP和LCP消耗时间过久，那么有可能是加载关键资源花的时间过久，也有可能是JavaScript执行过程中所花的时间过久，所以我们可以针对具体的情况来具体分析。

第三项是首屏时间(Speed Index)，这就是我们上面提到的LCP，它表示填满首屏页面所消耗的时间，首屏时间的值越大，那么加载速度越慢，具体的优化方式同优化第二项FMP是一样。

第四项是首次CPU空闲时间(First CPU Idle)，也称为First Interactive，它表示页面达到最小化可交互的时间，也就是说并不需要等到页面上的所有元素都可交互，只要可以对大部分用户输入做出响应即可。要缩短首次CPU空闲时长，我们就需要尽可能快地加载完关键资源，尽可能快地渲染出来首屏内容，因此优化方式和第二项FMP和第三项LCP是一样的。

第五项是完全可交互时间(Time to Interactive)，简称TTI，它表示页面中所有元素都达到了可交互的时长。简单理解就这时候页面的内容已经完全显示出来了，所有的JavaScript事件已经注册完成，页面能够对用户的交互做出快速响应，通常满足响应速度在50毫秒以内。如果要解决TTI时间过久的问题，我们可以推迟执行一些和生成页面无关的JavaScript工作。

第六项是最大估计输入延时(Max Potential First Input Delay），这个指标是估计你的Web页面在加载最繁忙的阶段， 窗口中响应用户输入所需的时间，为了改善该指标，我们可以使用WebWorker来执行一些计算，从而释放主线程。另一个有用的措施是重构CSS选择器，以确保它们执行较少的计算。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容：

本文我们主要讨论如何优化加载阶段的Web应用的性能。

要想优化Web性能，首先得需要有Web应用的性能数据。所以接下来，我们介绍了Chrome采集Web性能数据的两个工具：Performance和Audits，Performance可以采集非常多的性能，但是其使用难度大，相反，Audtis就简单了许多，它会分析检测到的性能数据并给出站点的性能得分，同时，还会给我们提供一些优化建议。

我们先从简单的工具上手，所以本文我们主要分析了Audits的使用方式，先介绍了如何使用Audits生成性能报告，然后我们解读了性能报告中的每一项内容。

大致了解Audits生成的性能报告之后，我们又分析Web应用在加载阶段的几个关键时间点，最后我们分析性能指标的具体含义以及如何提高性能指标的分数，从而达到优化Web应用的目的。

通过介绍，我们知道了Audits非常适合用来分析加载阶段的Web性能，除此之外，Audits还有其他非常实用的功能，比如可以检测我们的代码是否符合一些最佳实践，并给出提示，这样我们就可以根据Audits的提示来决定是否需要优化我们的代码，这个功能非常不错，具体使用方式留给你自己去摸索了。

课后思考

在文中我们又分析Web应用在加载阶段的几个关键时间点，在Audits中，通过对这些时间点的分析，输出了文中介绍的六项性能指标，其实这些时间点也可以通过Performance的时间线(Timelines)来查看，那么今天留给你的任务是：提前熟悉下Performance工具，并对照这文中加载阶段的几个时间点来熟悉下Performance的时间线(Timelines)，欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

你好，我是李兵。

作为一名前端工程师，除了需要编写功能性的代码以外，我们还需要关注Web应用的性能问题，我们应该有能力让我们的Web应用占用最小的资源，并以最高性能运行，这也是前端工程师进阶的必要能力。既然性能这么重要，那么我们今天要来聊聊Web性能问题。

到底什么是Web性能？

我们看下wiki对Web 性能的定义：

> Web 性能描述了Web应用在浏览器上的加载和显示的速度。

因此，当我们讨论Web性能时，其实就是讨论Web应用速度，关于Web应用的速度，我们需要从两个阶段来考虑：

- 页面加载阶段；
- 页面交互阶段。

在本文中，我们会将焦点放到第一个阶段：页面加载阶段的性能，在下篇文章中，我们会来重点分析页面交互阶段的性能。

性能检测工具：Performance vs Audits

要想优化Web的性能，我们就需要有监控Web应用的性能数据，那怎么监控呢？

如果没有工具来模拟各种不同的场景并统计各种性能指标，那么定位Web应用的性能瓶颈将是一件非常困难的任务。幸好，Chrome为我们提供了非常完善的性能检测工具：**Performance**和**Audits**，它们能够准确统计页面在加载阶段和运行阶段的一些核心数据，诸如任务执行记录、首屏展示花费的时长等，有了这些数据我们就能很容易定位到Web应用的**性能瓶颈** 。

首先Performance非常强大，因为它为我们提供了非常多的运行时数据，利用这些数据我们就可以分析出来Web应用的瓶颈。但是要完全学会其使用方式却是非常有难度的，其难点在于这些数据涉及到了特别多的概念，而这些概念又和浏览器的系统架构、消息循环机制、渲染流水线等知识紧密联系在了一起。

相反，Audtis就简单了许多，它将检测到的细节数据隐藏在背后，只提供给我们一些直观的性能数据，同时，还会给我们提供一些优化建议。

Perfomance能让我们看到更多细节数据，但是更加复杂，Audits就比较智能，但是隐藏了更多细节。为了能够让你循序渐进地理解内容，所以本节我们先从简单的Audits入手，看看如何利用它来检测和优化页面在加载阶段的性能，然后在下一节我们再来分析Perfomance。

检测之前准备工作

不过在检测Web的性能指标之前，我们还要配置好工作环境，具体地讲，你需要准备以下内容：

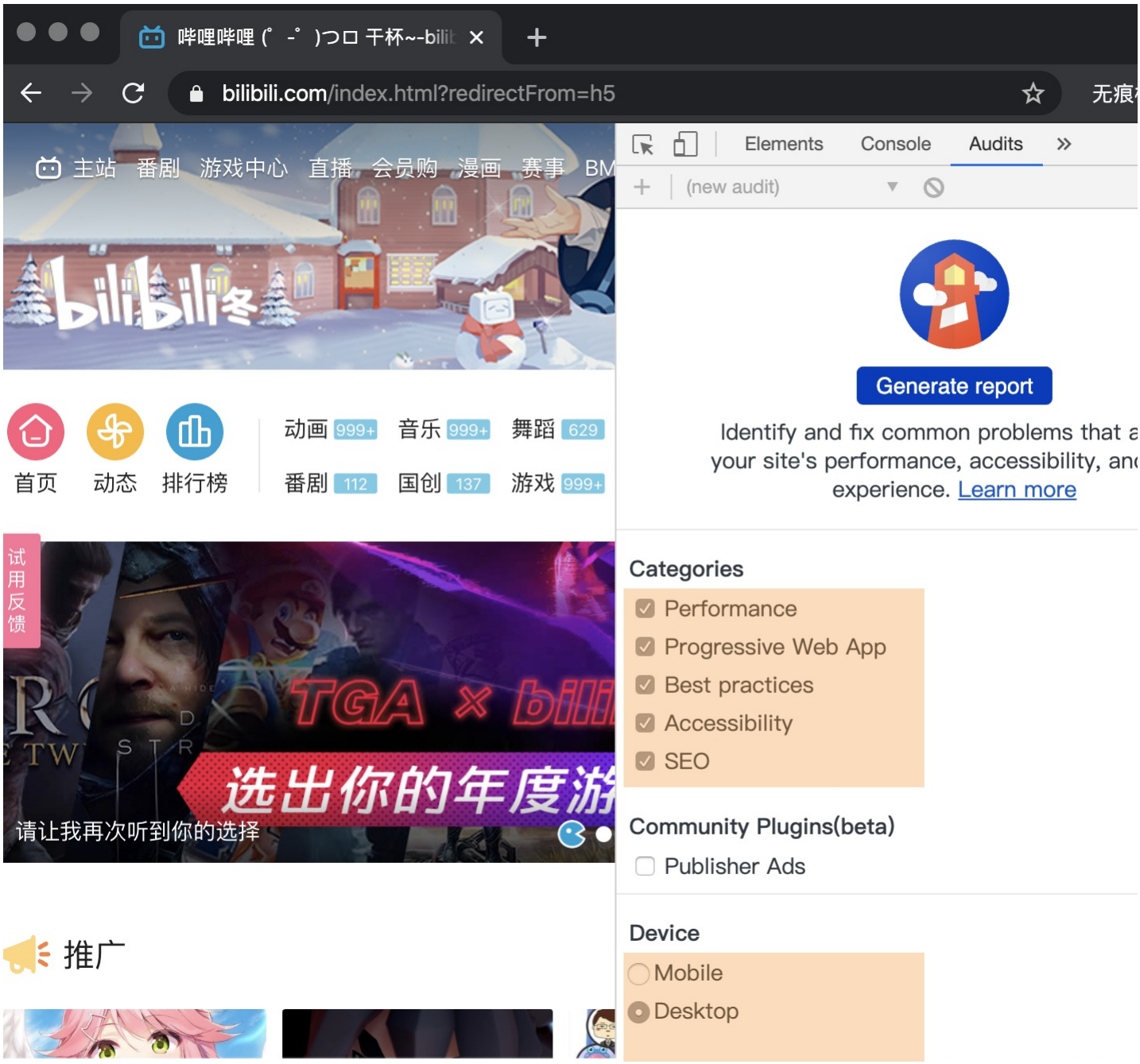
- 首先准备Chrome Canary版的浏览器，Chrome Canary是采用最新技术构建的，它的开发者工具和浏览器特性都是最新的，所以我推荐你使用Chrome Canary来做性能分析。当然你也可以使用稳定版的Chrome。
- 然后我们需要在Chrome的隐身模式下工作，这样可以确保我们安装的扩展、浏览器缓存、Cookie等数据不会影响到检测结果。

利用Audits生成Web性能报告

环境准备好了之后，我们就可以生成站点在加载阶段的性能报告了，这里我们可以拿B站作为分析的列子。

- 首先我们打开浏览器的隐身窗口，Windows系统下面的快捷键是Control+Shift+N，Mac系统下面的快捷键是Command+Shift+N。
- 然后在隐身窗口中输入B站的网站。
- 打开Chrome的开发者工具，选择Audits标签。

最终打开的页面如下图所示：



Audits界面

观察上图中的Audits界面，我们可以看到，在生成报告之前，我们需要先配置Audits，配置模块主要有两部分组成，一个是**监测类型(Categories)**，另外一个是**设备类型(Device)**。

监控类型(Categories)是指需要监控哪些内容，这里有五个对应的选项，它们的功能分别是：

- 监测并分析Web性能(**Performance**)；
- 监测并分析PWA(**Progressive Web App**)程序的性能；
- 监测并分析Web应用是否采用了最佳实践策略(**Best practices**)；
- 监测并分析是否实施了无障碍功能(**Accessibility**)，无障碍功能让一些身体有障碍的人可以方便地浏览你的Web应用。
- 监测并分析Web应用是否采实施了SEO搜索引擎优化(**SEO**)。

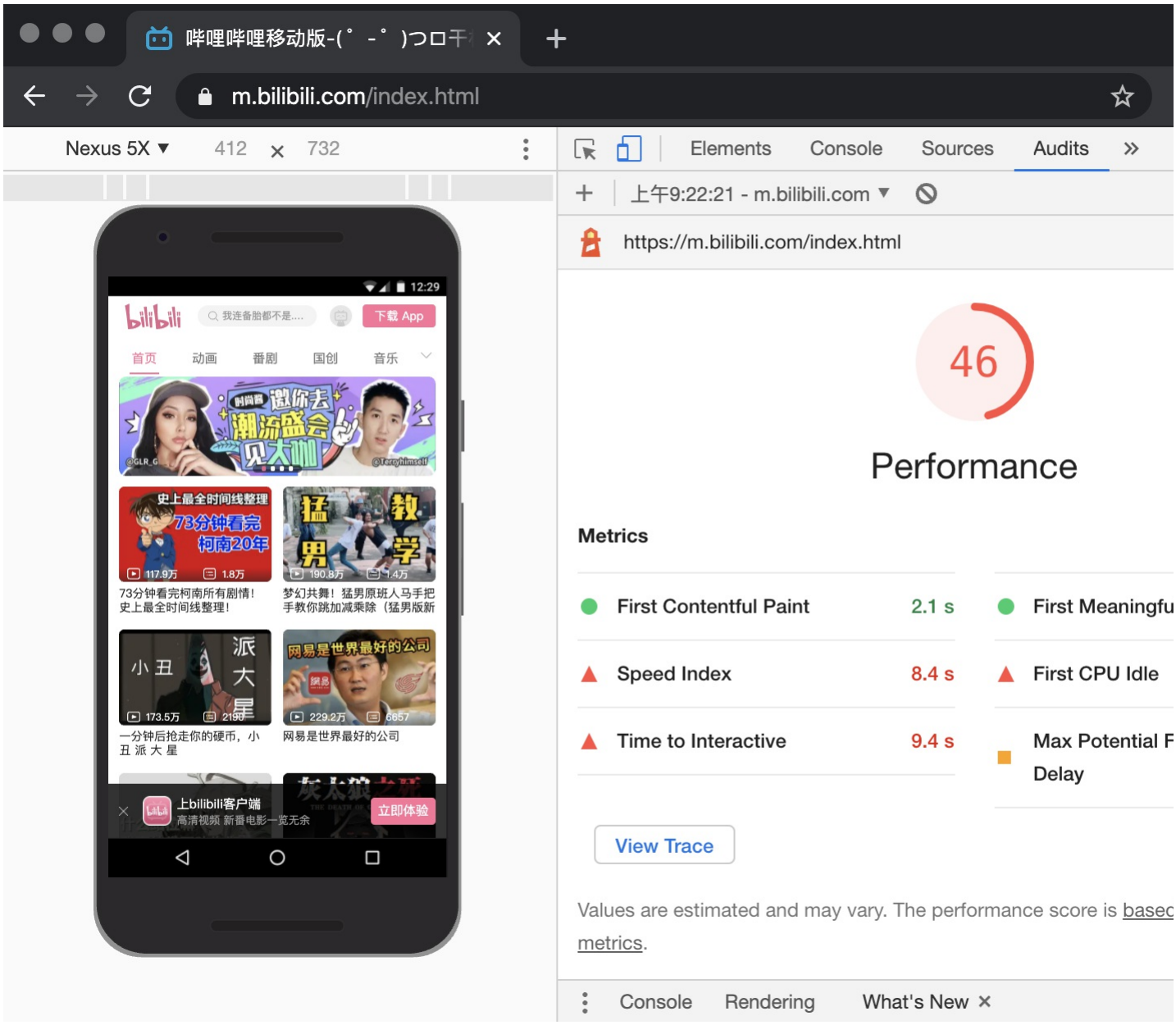
本文我们只需要关注Web应用的加载性能，所以勾选第一个Performance选项就可以了。

再看看**设备(Device)部分**，它给了我们两个选项，Moblie选项是用来模拟移动设备环境的，另外一个Desktop选项是用来模拟桌面环境的。这里我们选择移动设备选项，因为目前大多数流量都是由移动设备产生的，所以移动设备上的Web性能显得更加重要。

配置好选项之后，我们就可以点击最上面的生成报告(Generate report)按钮来生成报告了。

解读性能报告

点击生成报告的按钮之后，我们大约需要等待一分钟左右，Audits就可以生成最终的分析报告了，如下图所示：



生成的报告图

观察上图的分析报告，中间圆圈中的数字表示该站点在加载过程中的总体Web性能得分，总分是100分。我们目前的得分为46分，这表示该站点加载阶段的性能还有很大的提升空间。

Audits除了生成性能指标以外，还会分析该站点并提供了很多优化建议，我们可以根据这些建议来改进Web应用以获得更高的得分，进而获得更好的用户体验效果。

既能分析Web性能得分又能给出优化建议，所以Audits的分析报告还是非常有价值的，那么接下来，我们就来解读下Audits生成的性能报告。

报告的第一个部分是**性能指标(Metrics)**，如下图所示：

Metrics

First Contentful Paint	2.1 s	First Meaningful Paint	2.1 s
Speed Index	8.4 s	First CPU Idle	7.2 s
Time to Interactive	9.4 s	Max Potential First Input Delay	230 ms

View Trace

Values are estimated and may vary. The performance score is based only on these metrics.

性能指标

观察上图，我们可以发现性能指标下面一共有六项内容，这六项内容分别对应了从Web应用的加载到页面展示完成的这段时间中，各个阶段所消耗的时长。在中间还有一个View Trace按钮，点击该按钮可以跳转到Performance标签，并且查看这些阶段在Performance中所对应的位置。最下方是加载过程中各个时间段的屏幕截图。

报告的第二个部分是**可优化项(Opportunities)**，如下图所示：

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Eliminate render-blocking resources	0.76 s
Preconnect to required origins	0.38 s
Warnings: A preconnect <link> was found for "https://api.bilibili.com" but was not used by the browser. Check that you are using the `crossorigin` attribute properly.	
Properly size images	0.16 s

可优化项(Opportunities)

这些可优化项是Audits发现页面中的一些可以直接优化的部分，你可以对照Audits给的这些提示来优化你的Web应用。

报告的第三部分是**手动诊断(Diagnostics)**，如下图所示：

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- Avoid enormous network payloads — Total size was 4,257 KB
- Serve static assets with an efficient cache policy — 7 resources found
- Avoid an excessive DOM size — 892 elements
- Avoid chaining critical requests — 12 chains found
- Keep request counts low and transfer sizes small — 236 requests • 4,257 KB

Passed audits (16)

手动诊断(Diagnostics)

在手动诊断部分，采集了一些可能存在性能问题的指标，这些指标可能会影响到页面的加载性能，Audits把详情列出来，并让你依据实际情况，来手动排查每一项。

报告的最后一部分是**运行时设置(Runtime Settings)**，如下图所示：

Runtime Settings

URL	https://m.bilibili.com/index.html
Fetch time	Nov 30, 2019, 9:22 AM GMT+8
Device	Emulated Nexus 5X
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3980.0 Safari/537.36
User agent (network)	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Mobile Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1612

Generated by **Lighthouse** 5.7.0 | File an issue

运行时设置(Runtime Settings)

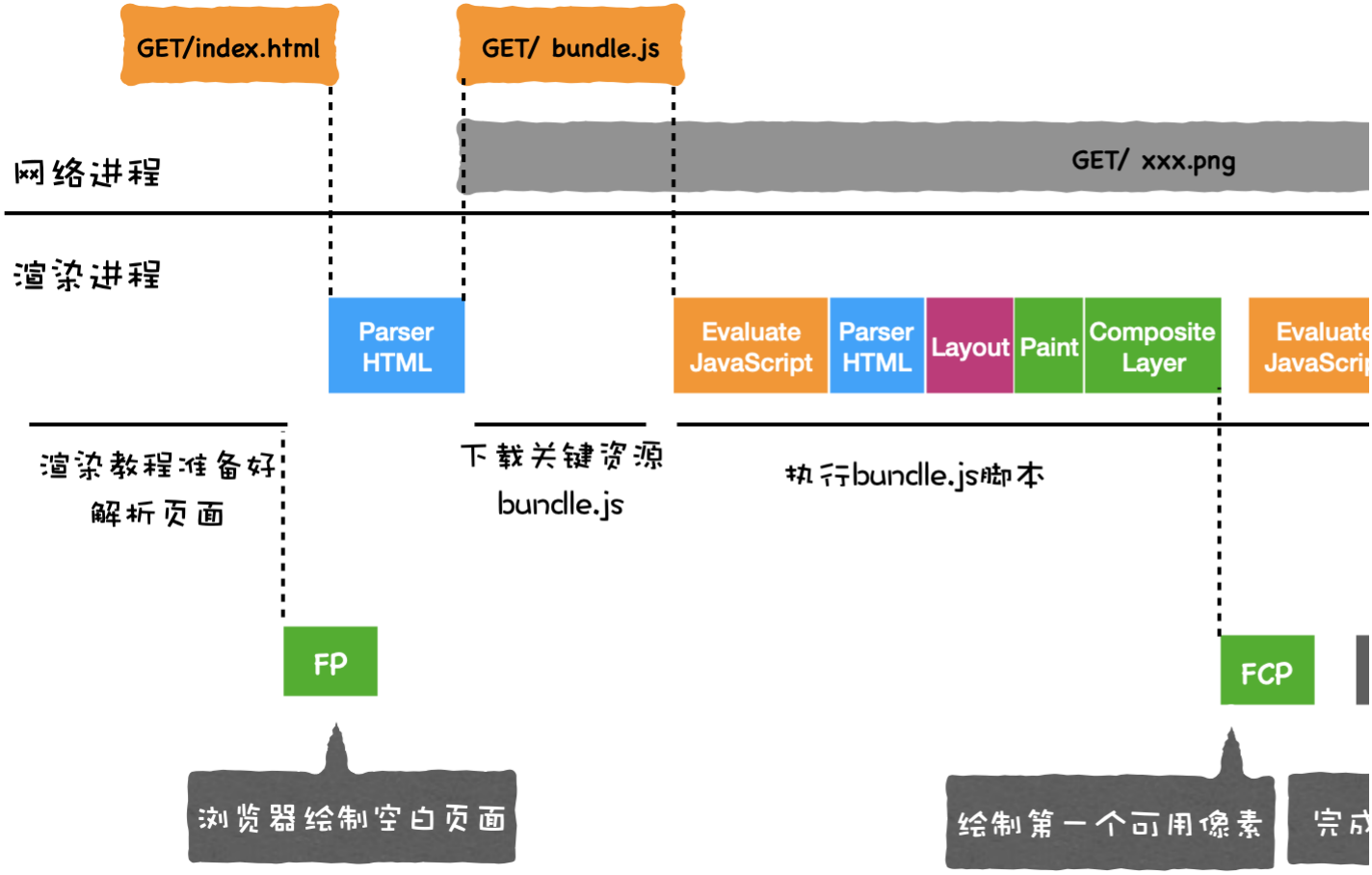
观察上图，这是运行时的一些基本数据，如果选择移动设备模式，你可以看到发送网络请求时的User Agent 会变成设备相关信息，还有会模拟设备的网速，这个体现在网络限速上。

根据性能报告优化Web性能

现在有了性能报告，接下来我们就可以依据报告来分析如何优化Web应用了。最直接的方式是想办法提高性能指标的分数，而性能指标的分数是由六项指标决定的，它们分别是：

1. 首次绘制(First Paint)；
2. 首次有效绘制(First Meaningfull Paint)；
3. 首屏时间(Speed Index)；
4. 首次CPU空闲时间(First CPU Idle)；
5. 完全可交互时间(Time to Interactive)；
6. 最大估计输入延时(Max Potential First Input Delay)。

那么接下来我会逐一分析六项指标的含义，并讨论如何提升这六项指标的数值。这六项都是页面在加载过程中的性能指标，所以要弄明白这六项指标的具体含义，我们还得结合页面的加载过程来分析。一图胜过千言，我们还是先看下面这张页面从加载到展示的过程图：



页面加载过程

观察上图的页面加载过程，我们发现，在渲染进程确认要渲染当前的请求后，渲染进程会创建一个空白页面，我们把创建空白页面的这个时间点称为**First Paint**，简称**FP**。

然后渲染进程继续请求关键资源，我们在《25丨页面性能：如何系统地优化页面？》这节中介绍过了关键资源，并且知道了关键资源包括了JavaScript文件和CSS文件，因为关键资源会阻塞页面的渲染，所以我们需要等待关键资源加载完成后，才能执行进一步的页面绘制。

上图中，bundle.js是关键资源，因此需要完成加载之后，渲染进程才能执行该脚本，然后脚本会修改DOM，引发重绘和重排等一系列操作，当页面中绘制了第一个像素时，我们把这个时间点称为**First Content Paint**，简称**FCP**。

接下来继续执行JavaScript脚本，当首屏内容完全绘制完成时，我们把这个时间点称为**Largest Content Paint**，简称**LCP**。

在FCP和LCP中间，还有一个FMP，这个是首次有效绘制，由于FMP计算复杂，而且容易出错，现在不推荐使用该指标，所以这里我们也不做过多介绍了。

接下来JavaScript脚本执行结束，渲染进程判断该页面的DOM生成完毕，于是触发DOMContentLoad事件。等所有资源都加载结束之后，再触发onload事件。

好了，以上就是页面在加载过程中各个重要的时间节点，了解了这些时间节点，我们就可以来聊聊性能报告的六项指标的含义并讨论如何优化这些指标。

我们先来分析下**第一项指标FP**，如果FP时间过久，那么直接说明了一个问题，那就是页面的HTML文件可能由于网络原因导致加载时间过久，这块具体的分析过程你可以参考《21丨Chrome开发者工具：利用网络面板做性能分析》这节内容。

第二项是FMP，上面也提到过由于FMP计算复杂，所以现在不建议使用该指标了，另外由于LCP的计算规则简单，所以推荐使用LCP指标，具体文章你可以参考这里。不过是FMP还是LCP，优化它们的方式都是类似的，你可以结合上图，如果FMP和LCP消耗时间过久，那么有可能是加载关键资源花的时间过久，也有可能是JavaScript执行过程中所花的时间过久，所以我们可以针对具体的情况来具体分析。

第三项是首屏时间(Speed Index)，这就是我们上面提到的LCP，它表示填满首屏页面所消耗的时间，首屏时间的值越大，那么加载速度越慢，具体的优化方式同优化第二项FMP是一样。

第四项是首次CPU空闲时间(First CPU Idle)，也称为First Interactive，它表示页面达到最小化可交互的时间，也就是说并不需要等到页面上的所有元素都可交互，只要可以对大部分用户输入做出响应即可。要缩短首次CPU空闲时长，我们就需要尽可能快地加载完关键资源，尽可能快地渲染出来首屏内容，因此优化方式和第二项FMP和第三项LCP是一样的。

第五项是完全可交互时间(Time to Interactive)，简称TTI，它表示页面中所有元素都达到了可交互的时长。简单理解就这时候页面的内容已经完全显示出来了，所有的JavaScript事件已经注册完成，页面能够对用户的交互做出快速响应，通常满足响应速度在50毫秒以内。如果要解决TTI时间过久的问题，我们可以推迟执行一些和生成页面无关的JavaScript工作。

第六项是最大估计输入延时(Max Potential First Input Delay），这个指标是估计你的Web页面在加载最繁忙的阶段， 窗口中响应用户输入所需的时间，为了改善该指标，我们可以使用WebWorker来执行一些计算，从而释放主线程。另一个有用的措施是重构CSS选择器，以确保它们执行较少的计算。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容：

本文我们主要讨论如何优化加载阶段的Web应用的性能。

要想优化Web性能，首先得需要有Web应用的性能数据。所以接下来，我们介绍了Chrome采集Web性能数据的两个工具：Performance和Audits，Performance可以采集非常多的性能，但是其使用难度大，相反，Audtis就简单了许多，它会分析检测到的性能数据并给出站点的性能得分，同时，还会给我们提供一些优化建议。

我们先从简单的工具上手，所以本文我们主要分析了Audits的使用方式，先介绍了如何使用Audits生成性能报告，然后我们解读了性能报告中的每一项内容。

大致了解Audits生成的性能报告之后，我们又分析Web应用在加载阶段的几个关键时间点，最后我们分析性能指标的具体含义以及如何提高性能指标的分数，从而达到优化Web应用的目的。

通过介绍，我们知道了Audits非常适合用来分析加载阶段的Web性能，除此之外，Audits还有其他非常实用的功能，比如可以检测我们的代码是否符合一些最佳实践，并给出提示，这样我们就可以根据Audits的提示来决定是否需要优化我们的代码，这个功能非常不错，具体使用方式留给你自己去摸索了。

课后思考

在文中我们又分析Web应用在加载阶段的几个关键时间点，在Audits中，通过对这些时间点的分析，输出了文中介绍的六项性能指标，其实这些时间点也可以通过Performance的时间线(Timelines)来查看，那么今天留给你的任务是：提前熟悉下Performance工具，并对照这文中加载阶段的几个时间点来熟悉下Performance的时间线(Timelines)，欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

你好，我是李兵。

作为一名前端工程师，除了需要编写功能性的代码以外，我们还需要关注Web应用的性能问题，我们应该有能力让我们的Web应用占用最小的资源，并以最高性能运行，这也是前端工程师进阶的必要能力。既然性能这么重要，那么我们今天要来聊聊Web性能问题。

到底什么是Web性能？

我们看下wiki对Web 性能的定义：

> Web 性能描述了Web应用在浏览器上的加载和显示的速度。

因此，当我们讨论Web性能时，其实就是讨论Web应用速度，关于Web应用的速度，我们需要从两个阶段来考虑：

- 页面加载阶段；
- 页面交互阶段。

在本文中，我们会将焦点放到第一个阶段：页面加载阶段的性能，在下篇文章中，我们会来重点分析页面交互阶段的性能。

性能检测工具：Performance vs Audits

要想优化Web的性能，我们就需要有监控Web应用的性能数据，那怎么监控呢？

如果没有工具来模拟各种不同的场景并统计各种性能指标，那么定位Web应用的性能瓶颈将是一件非常困难的任务。幸好，Chrome为我们提供了非常完善的性能检测工具：**Performance**和**Audits**，它们能够准确统计页面在加载阶段和运行阶段的一些核心数据，诸如任务执行记录、首屏展示花费的时长等，有了这些数据我们就能很容易定位到Web应用的**性能瓶颈** 。

首先Performance非常强大，因为它为我们提供了非常多的运行时数据，利用这些数据我们就可以分析出来Web应用的瓶颈。但是要完全学会其使用方式却是非常有难度的，其难点在于这些数据涉及到了特别多的概念，而这些概念又和浏览器的系统架构、消息循环机制、渲染流水线等知识紧密联系在了一起。

相反，Audtis就简单了许多，它将检测到的细节数据隐藏在背后，只提供给我们一些直观的性能数据，同时，还会给我们提供一些优化建议。

Perfomance能让我们看到更多细节数据，但是更加复杂，Audits就比较智能，但是隐藏了更多细节。为了能够让你循序渐进地理解内容，所以本节我们先从简单的Audits入手，看看如何利用它来检测和优化页面在加载阶段的性能，然后在下一节我们再来分析Perfomance。

检测之前准备工作

不过在检测Web的性能指标之前，我们还要配置好工作环境，具体地讲，你需要准备以下内容：

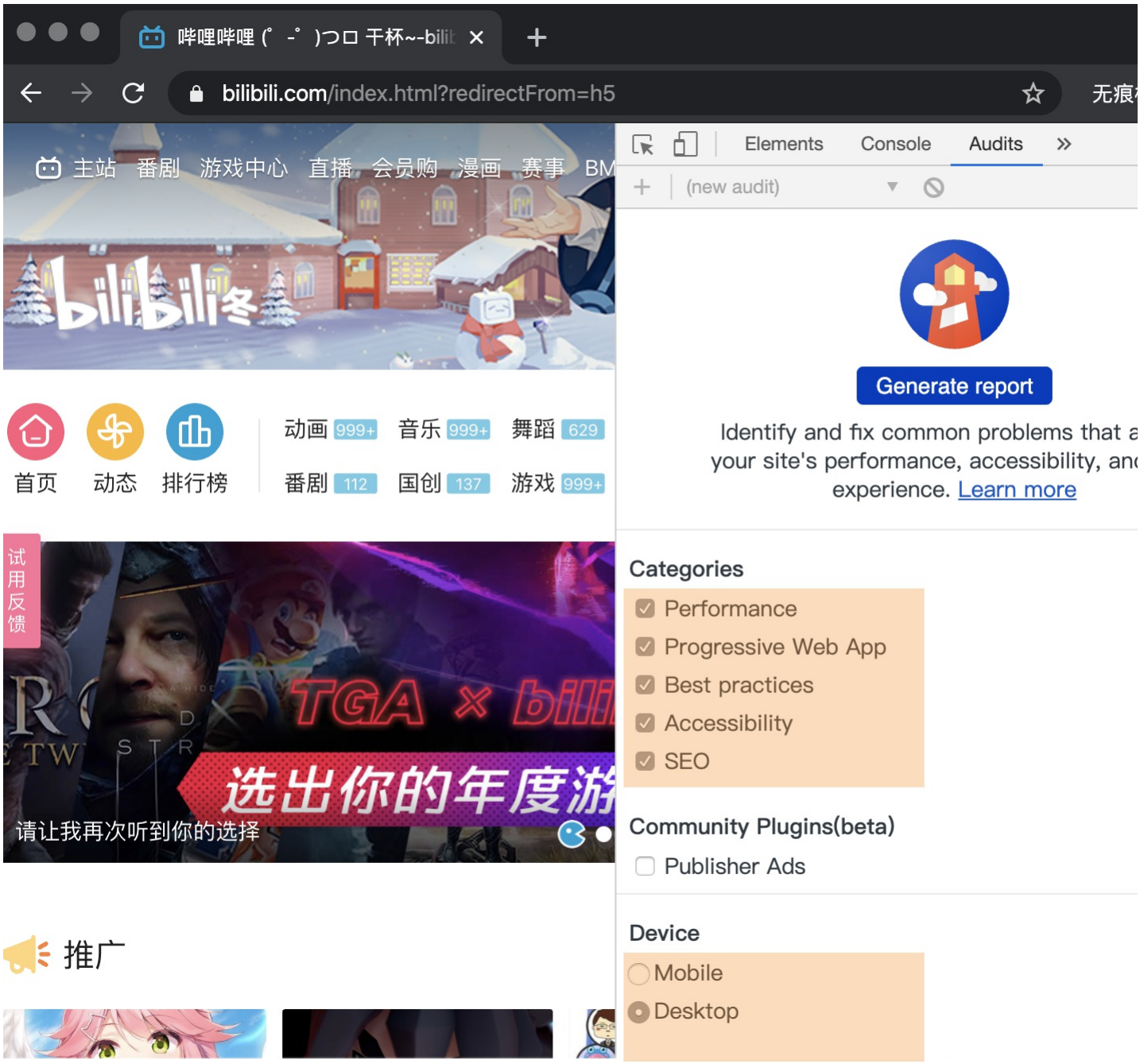
- 首先准备Chrome Canary版的浏览器，Chrome Canary是采用最新技术构建的，它的开发者工具和浏览器特性都是最新的，所以我推荐你使用Chrome Canary来做性能分析。当然你也可以使用稳定版的Chrome。
- 然后我们需要在Chrome的隐身模式下工作，这样可以确保我们安装的扩展、浏览器缓存、Cookie等数据不会影响到检测结果。

利用Audits生成Web性能报告

环境准备好了之后，我们就可以生成站点在加载阶段的性能报告了，这里我们可以拿B站作为分析的列子。

- 首先我们打开浏览器的隐身窗口，Windows系统下面的快捷键是Control+Shift+N，Mac系统下面的快捷键是Command+Shift+N。
- 然后在隐身窗口中输入B站的网站。
- 打开Chrome的开发者工具，选择Audits标签。

最终打开的页面如下图所示：



Audits界面

观察上图中的Audits界面，我们可以看到，在生成报告之前，我们需要先配置Audits，配置模块主要有两部分组成，一个是**监测类型(Categories)**，另外一个是**设备类型(Device)**。

监控类型(Categories)是指需要监控哪些内容，这里有五个对应的选项，它们的功能分别是：

- 监测并分析Web性能(**Performance**)；
- 监测并分析PWA(**Progressive Web App**)程序的性能；
- 监测并分析Web应用是否采用了最佳实践策略(**Best practices**)；
- 监测并分析是否实施了无障碍功能(**Accessibility**)，无障碍功能让一些身体有障碍的人可以方便地浏览你的Web应用。
- 监测并分析Web应用是否采实施了SEO搜索引擎优化(**SEO**)。

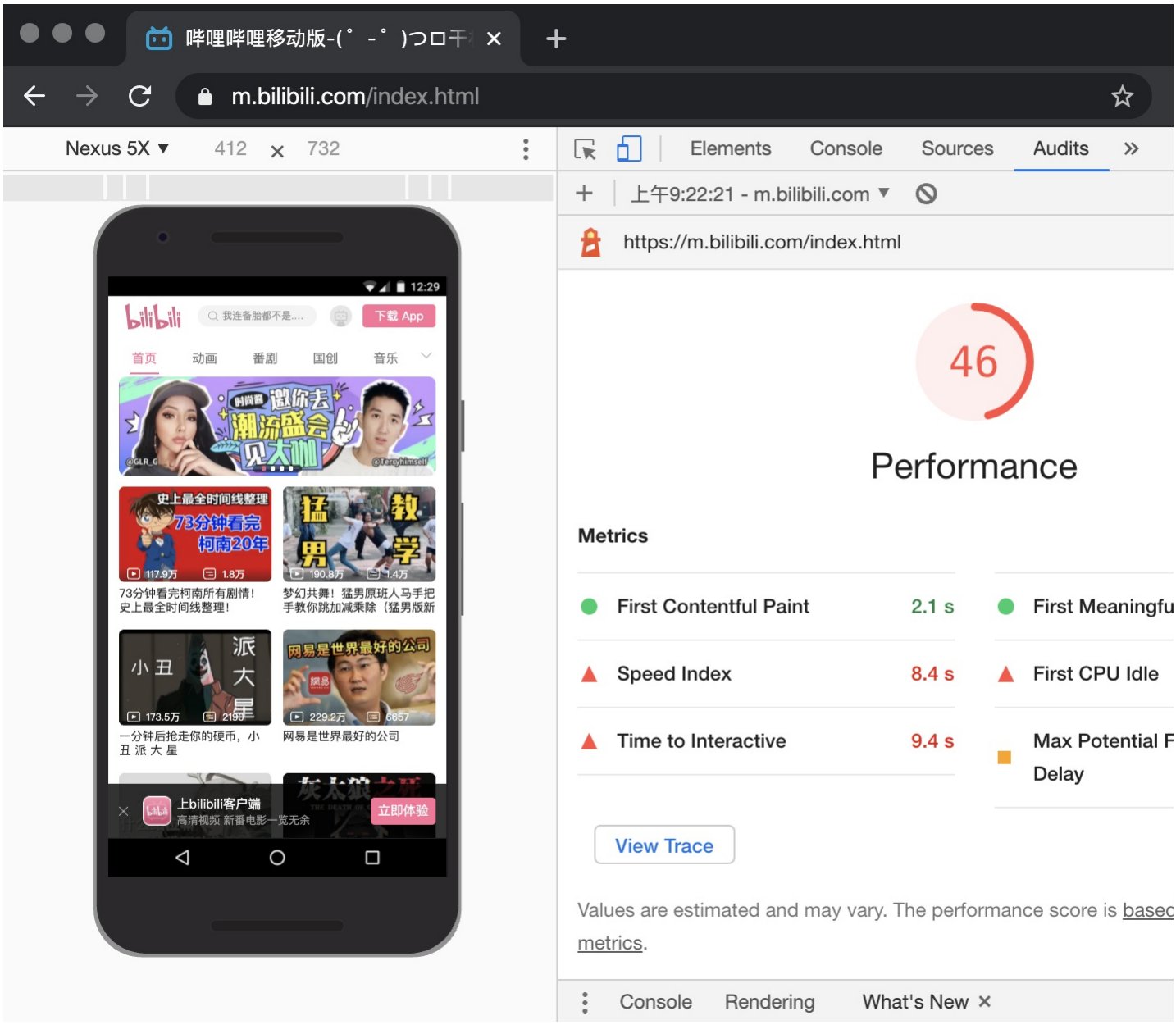
本文我们只需要关注Web应用的加载性能，所以勾选第一个Performance选项就可以了。

再看看**设备(Device)部分**，它给了我们两个选项，Moblie选项是用来模拟移动设备环境的，另外一个Desktop选项是用来模拟桌面环境的。这里我们选择移动设备选项，因为目前大多数流量都是由移动设备产生的，所以移动设备上的Web性能显得更加重要。

配置好选项之后，我们就可以点击最上面的生成报告(Generate report)按钮来生成报告了。

解读性能报告

点击生成报告的按钮之后，我们大约需要等待一分钟左右，Audits就可以生成最终的分析报告了，如下图所示：



生成的报告图

观察上图的分析报告，中间圆圈中的数字表示该站点在加载过程中的总体Web性能得分，总分是100分。我们目前的得分为46分，这表示该站点加载阶段的性能还有很大的提升空间。

Audits除了生成性能指标以外，还会分析该站点并提供了很多优化建议，我们可以根据这些建议来改进Web应用以获得更高的得分，进而获得更好的用户体验效果。

既能分析Web性能得分又能给出优化建议，所以Audits的分析报告还是非常有价值的，那么接下来，我们就来解读下Audits生成的性能报告。

报告的第一个部分是**性能指标(Metrics)**，如下图所示：

Metrics

First Contentful Paint	2.1 s	First Meaningful Paint	2.1 s
Speed Index	8.4 s	First CPU Idle	7.2 s
Time to Interactive	9.4 s	Max Potential First Input Delay	230 ms

View Trace

Values are estimated and may vary. The performance score is based only on these metrics.

性能指标

观察上图，我们可以发现性能指标下面一共有六项内容，这六项内容分别对应了从Web应用的加载到页面展示完成的这段时间中，各个阶段所消耗的时长。在中间还有一个View Trace按钮，点击该按钮可以跳转到Performance标签，并且查看这些阶段在Performance中所对应的位置。最下方是加载过程中各个时间段的屏幕截图。

报告的第二个部分是**可优化项(Opportunities)**，如下图所示：

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Eliminate render-blocking resources	0.76 s
Preconnect to required origins	0.38 s
Warnings: A preconnect <link> was found for "https://api.bilibili.com" but was not used by the browser. Check that you are using the `crossorigin` attribute properly.	
Properly size images	0.16 s

可优化项(Opportunities)

这些可优化项是Audits发现页面中的一些可以直接优化的部分，你可以对照Audits给的这些提示来优化你的Web应用。

报告的第三部分是**手动诊断(Diagnostics)**，如下图所示：

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- Avoid enormous network payloads — Total size was 4,257 KB
- Serve static assets with an efficient cache policy — 7 resources found
- Avoid an excessive DOM size — 892 elements
- Avoid chaining critical requests — 12 chains found
- Keep request counts low and transfer sizes small — 236 requests • 4,257 KB

Passed audits (16)

手动诊断(Diagnostics)

在手动诊断部分，采集了一些可能存在性能问题的指标，这些指标可能会影响到页面的加载性能，Audits把详情列出来，并让你依据实际情况，来手动排查每一项。

报告的最后一部分是**运行时设置(Runtime Settings)**，如下图所示：

Runtime Settings

URL	https://m.bilibili.com/index.html
Fetch time	Nov 30, 2019, 9:22 AM GMT+8
Device	Emulated Nexus 5X
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3980.0 Safari/537.36
User agent (network)	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Mobile Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1612

Generated by **Lighthouse** 5.7.0 | File an issue

运行时设置(Runtime Settings)

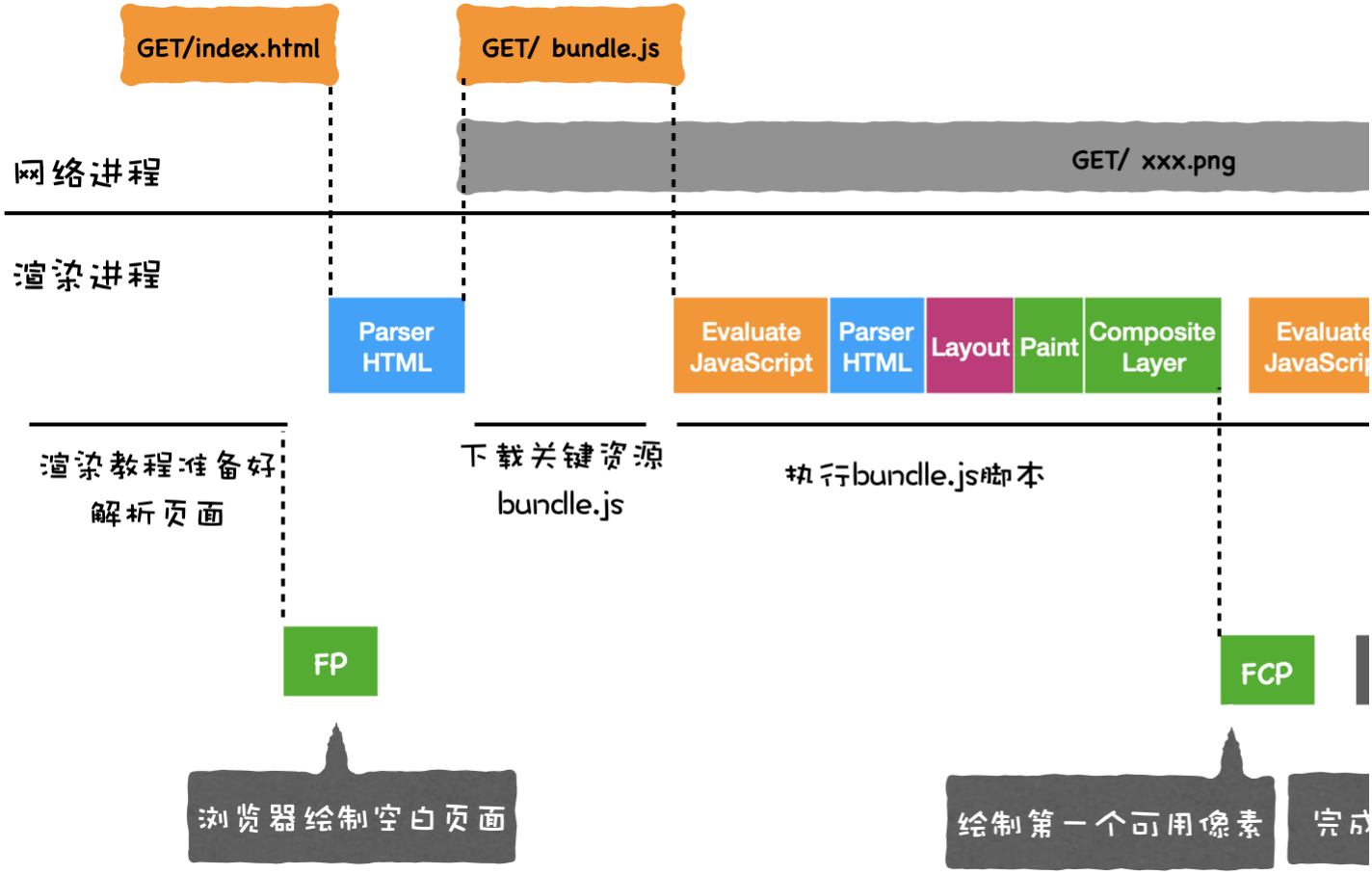
观察上图，这是运行时的一些基本数据，如果选择移动设备模式，你可以看到发送网络请求时的User Agent 会变成设备相关信息，还有会模拟设备的网速，这个体现在网络限速上。

根据性能报告优化Web性能

现在有了性能报告，接下来我们就可以依据报告来分析如何优化Web应用了。最直接的方式是想办法提高性能指标的分数，而性能指标的分数是由六项指标决定的，它们分别是：

1. 首次绘制(First Paint)；
2. 首次有效绘制(First Meaningfull Paint)；
3. 首屏时间(Speed Index)；
4. 首次CPU空闲时间(First CPU Idle)；
5. 完全可交互时间(Time to Interactive)；
6. 最大估计输入延时(Max Potential First Input Delay)。

那么接下来我会逐一分析六项指标的含义，并讨论如何提升这六项指标的数值。这六项都是页面在加载过程中的性能指标，所以要弄明白这六项指标的具体含义，我们还得结合页面的加载过程来分析。一图胜过千言，我们还是先看下面这张页面从加载到展示的过程图：



页面加载过程

观察上图的页面加载过程，我们发现，在渲染进程确认要渲染当前的请求后，渲染进程会创建一个空白页面，我们把创建空白页面的这个时间点称为**First Paint**，简称**FP**。

然后渲染进程继续请求关键资源，我们在《25 | 页面性能：如何系统地优化页面？》这节中介绍过了关键资源，并且知道了关键资源包括了JavaScript文件和CSS文件，因为关键资源会阻塞页面的渲染，所以我们需要等待关键资源加载完成后，才能执行进一步的页面绘制。

上图中，bundle.js是关键资源，因此需要完成加载之后，渲染进程才能执行该脚本，然后脚本会修改DOM，引发重绘和重排等一系列操作，当页面中绘制了第一个像素时，我们把这个时间点称为**First Content Paint**，简称**FCP**。

接下来继续执行JavaScript脚本，当首屏内容完全绘制完成时，我们把这个时间点称为**Largest Content Paint**，简称**LCP**。

在FCP和LCP中间，还有一个FMP，这个是首次有效绘制，由于FMP计算复杂，而且容易出错，现在不推荐使用该指标，所以这里我们也不做过多介绍了。

接下来JavaScript脚本执行结束，渲染进程判断该页面的DOM生成完毕，于是触发DOMContentLoad事件。等所有资源都加载结束之后，再触发onload事件。

好了，以上就是页面在加载过程中各个重要的时间节点，了解了这些时间节点，我们就可以来聊聊性能报告的六项指标的含义并讨论如何优化这些指标。

我们先来分析下**第一项指标FP**，如果FP时间过久，那么直接说明了一个问题，那就是页面的HTML文件可能由于网络原因导致加载时间过久，这块具体的分析过程你可以参考《21 | Chrome开发者工具：利用网络面板做性能分析》这节内容。

第二项是FMP，上面也提到过由于FMP计算复杂，所以现在不建议使用该指标了，另外由于LCP的计算规则简单，所以推荐使用LCP指标，具体文章你可以参考这里。不过是FMP还是LCP，优化它们的方式都是类似的，你可以结合上图，如果FMP和LCP消耗时间过久，那么有可能是加载关键资源花的时间过久，也有可能是JavaScript执行过程中所花的时间过久，所以我们可以针对具体的情况来具体分析。

第三项是首屏时间(Speed Index)，这就是我们上面提到的LCP，它表示填满首屏页面所消耗的时间，首屏时间的值越大，那么加载速度越慢，具体的优化方式同优化第二项FMP是一样。

第四项是首次CPU空闲时间(First CPU Idle)，也称为First Interactive，它表示页面达到最小化可交互的时间，也就是说并不需要等到页面上的所有元素都可交互，只要可以对大部分用户输入做出响应即可。要缩短首次CPU空闲时长，我们就需要尽可能快地加载完关键资源，尽可能快地渲染出来首屏内容，因此优化方式和第二项FMP和第三项LCP是一样的。

第五项是完全可交互时间(Time to Interactive)，简称TTI，它表示页面中所有元素都达到了可交互的时长。简单理解就这时候页面的内容已经完全显示出来了，所有的JavaScript事件已经注册完成，页面能够对用户的交互做出快速响应，通常满足响应速度在50毫秒以内。如果要解决TTI时间过久的问题，我们可以推迟执行一些和生成页面无关的JavaScript工作。

第六项是最大估计输入延时(Max Potential First Input Delay），这个指标是估计你的Web页面在加载最繁忙的阶段， 窗口中响应用户输入所需的时间，为了改善该指标，我们可以使用WebWorker来执行一些计算，从而释放主线程。另一个有用的措施是重构CSS选择器，以确保它们执行较少的计算。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容：

本文我们主要讨论如何优化加载阶段的Web应用的性能。

要想优化Web性能，首先得需要有Web应用的性能数据。所以接下来，我们介绍了Chrome采集Web性能数据的两个工具：Performance和Audits，Performance可以采集非常多的性能，但是其使用难度大，相反，Audtis就简单了许多，它会分析检测到的性能数据并给出站点的性能得分，同时，还会给我们提供一些优化建议。

我们先从简单的工具上手，所以本文我们主要分析了Audits的使用方式，先介绍了如何使用Audits生成性能报告，然后我们解读了性能报告中的每一项内容。

大致了解Audits生成的性能报告之后，我们又分析Web应用在加载阶段的几个关键时间点，最后我们分析性能指标的具体含义以及如何提高性能指标的分数，从而达到优化Web应用的目的。

通过介绍，我们知道了Audits非常适合用来分析加载阶段的Web性能，除此之外，Audits还有其他非常实用的功能，比如可以检测我们的代码是否符合一些最佳实践，并给出提示，这样我们就可以根据Audits的提示来决定是否需要优化我们的代码，这个功能非常不错，具体使用方式留给你自己去摸索了。

课后思考

在文中我们又分析Web应用在加载阶段的几个关键时间点，在Audits中，通过对这些时间点的分析，输出了文中介绍的六项性能指标，其实这些时间点也可以通过Performance的时间线(Timelines)来查看，那么今天留给你的任务是：提前熟悉下Performance工具，并对照这文中加载阶段的几个时间点来熟悉下Performance的时间线(Timelines)，欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

你好，我是李兵。

作为一名前端工程师，除了需要编写功能性的代码以外，我们还需要关注Web应用的性能问题，我们应该有能力让我们的Web应用占用最小的资源，并以最高性能运行，这也是前端工程师进阶的必要能力。既然性能这么重要，那么我们今天要来聊聊Web性能问题。

到底什么是Web性能？

我们看下wiki对Web 性能的定义：

> Web 性能描述了Web应用在浏览器上的加载和显示的速度。

因此，当我们讨论Web性能时，其实就是讨论Web应用速度，关于Web应用的速度，我们需要从两个阶段来考虑：

- 页面加载阶段；
- 页面交互阶段。

在本文中，我们会将焦点放到第一个阶段：页面加载阶段的性能，在下篇文章中，我们会来重点分析页面交互阶段的性能。

性能检测工具：Performance vs Audits

要想优化Web的性能，我们就需要有监控Web应用的性能数据，那怎么监控呢？

如果没有工具来模拟各种不同的场景并统计各种性能指标，那么定位Web应用的性能瓶颈将是一件非常困难的任务。幸好，Chrome为我们提供了非常完善的性能检测工具：**Performance**和**Audits**，它们能够准确统计页面在加载阶段和运行阶段的一些核心数据，诸如任务执行记录、首屏展示花费的时长等，有了这些数据我们就能很容易定位到Web应用的**性能瓶颈** 。

首先Performance非常强大，因为它为我们提供了非常多的运行时数据，利用这些数据我们就可以分析出来Web应用的瓶颈。但是要完全学会其使用方式却是非常有难度的，其难点在于这些数据涉及到了特别多的概念，而这些概念又和浏览器的系统架构、消息循环机制、渲染流水线等知识紧密联系在了一起。

相反，Audtis就简单了许多，它将检测到的细节数据隐藏在背后，只提供给我们一些直观的性能数据，同时，还会给我们提供一些优化建议。

Perfomance能让我们看到更多细节数据，但是更加复杂，Audits就比较智能，但是隐藏了更多细节。为了能够让你循序渐进地理解内容，所以本节我们先从简单的Audits入手，看看如何利用它来检测和优化页面在加载阶段的性能，然后在下一节我们再来分析Perfomance。

检测之前准备工作

不过在检测Web的性能指标之前，我们还要配置好工作环境，具体地讲，你需要准备以下内容：

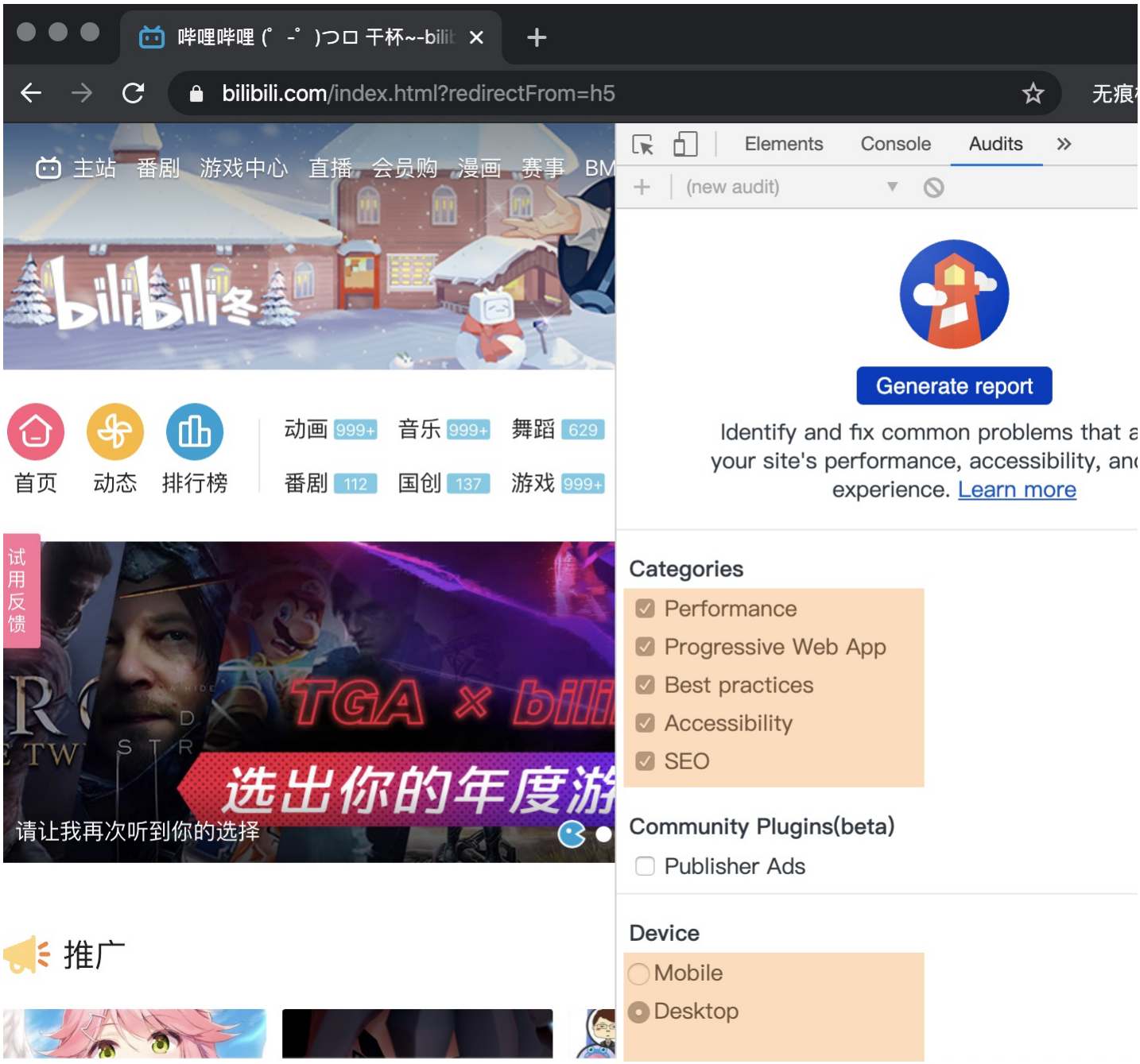
- 首先准备Chrome Canary版的浏览器，Chrome Canary是采用最新技术构建的，它的开发者工具和浏览器特性都是最新的，所以我推荐你使用Chrome Canary来做性能分析。当然你也可以使用稳定版的Chrome。
- 然后我们需要在Chrome的隐身模式下工作，这样可以确保我们安装的扩展、浏览器缓存、Cookie等数据不会影响到检测结果。

利用Audits生成Web性能报告

环境准备好了之后，我们就可以生成站点在加载阶段的性能报告了，这里我们可以拿B站作为分析的列子。

- 首先我们打开浏览器的隐身窗口，Windows系统下面的快捷键是Control+Shift+N，Mac系统下面的快捷键是Command+Shift+N。
- 然后在隐身窗口中输入B站的网站。
- 打开Chrome的开发者工具，选择Audits标签。

最终打开的页面如下图所示：



Audits界面

观察上图中的Audits界面，我们可以看到，在生成报告之前，我们需要先配置Audits，配置模块主要有两部分组成，一个是**监测类型(Categories)**，另外一个是**设备类型(Device)**。

监控类型(Categories)是指需要监控哪些内容，这里有五个对应的选项，它们的功能分别是：

- 监测并分析Web性能(**Performance**)；
- 监测并分析PWA(**Progressive Web App**)程序的性能；
- 监测并分析Web应用是否采用了最佳实践策略(**Best practices**)；
- 监测并分析是否实施了无障碍功能(**Accessibility**)，无障碍功能让一些身体有障碍的人可以方便地浏览你的Web应用。
- 监测并分析Web应用是否采实施了SEO搜索引擎优化(**SEO**)。

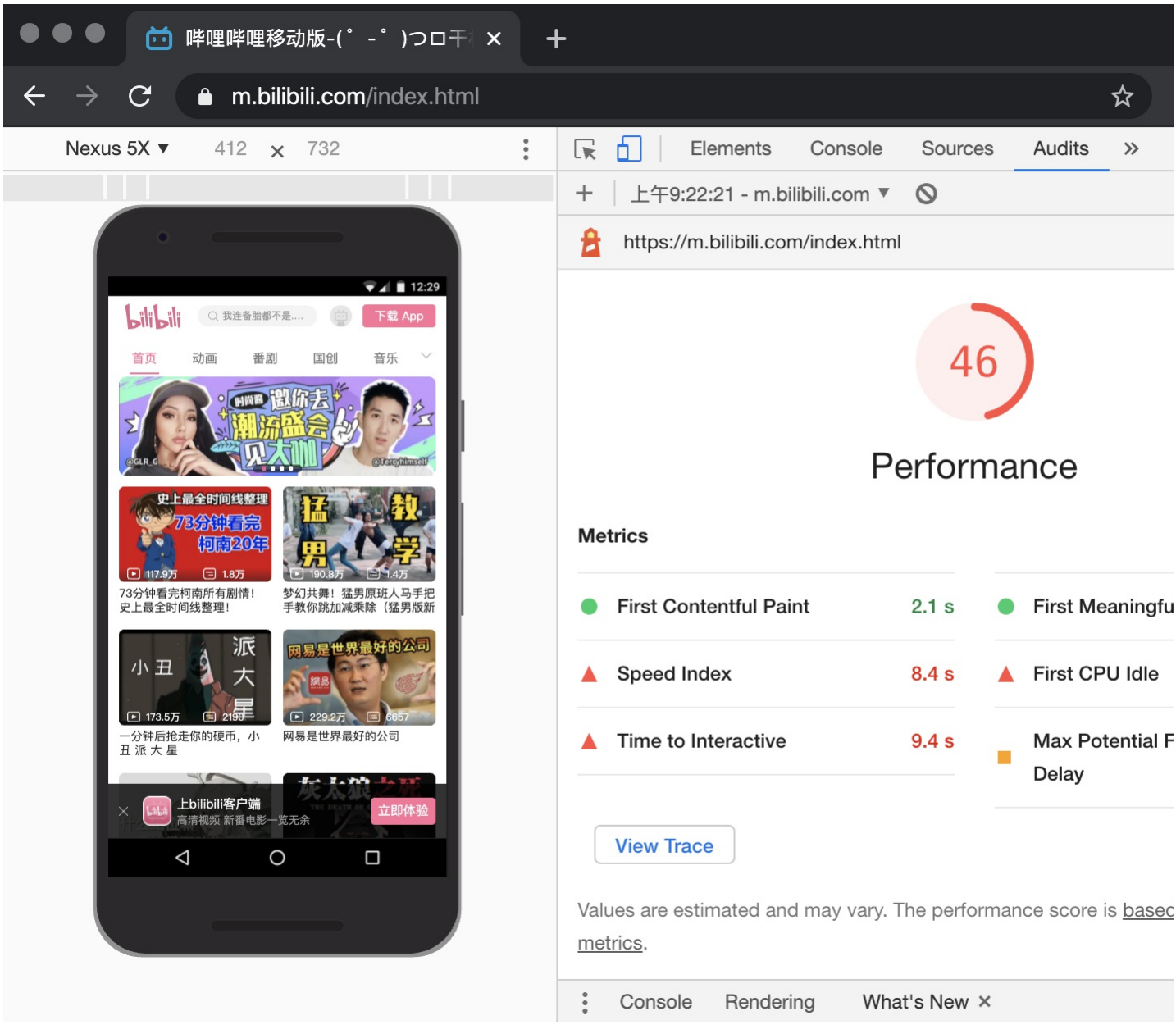
本文我们只需要关注Web应用的加载性能，所以勾选第一个Performance选项就可以了。

再看看**设备(Device)部分**，它给了我们两个选项，Moblie选项是用来模拟移动设备环境的，另外一个Desktop选项是用来模拟桌面环境的。这里我们选择移动设备选项，因为目前大多数流量都是由移动设备产生的，所以移动设备上的Web性能显得更加重要。

配置好选项之后，我们就可以点击最上面的生成报告(Generate report)按钮来生成报告了。

解读性能报告

点击生成报告的按钮之后，我们大约需要等待一分钟左右，Audits就可以生成最终的分析报告了，如下图所示：



生成的报告图

观察上图的分析报告，中间圆圈中的数字表示该站点在加载过程中的总体Web性能得分，总分是100分。我们目前的得分为46分，这表示该站点加载阶段的性能还有很大的提升空间。

Audits除了生成性能指标以外，还会分析该站点并提供了很多优化建议，我们可以根据这些建议来改进Web应用以获得更高的得分，进而获得更好的用户体验效果。

既能分析Web性能得分又能给出优化建议，所以Audits的分析报告还是非常有价值的，那么接下来，我们就来解读下Audits生成的性能报告。

报告的第一个部分是**性能指标(Metrics)**，如下图所示：

Metrics

First Contentful Paint	2.1 s	First Meaningful Paint	2.1 s
Speed Index	8.4 s	First CPU Idle	7.2 s
Time to Interactive	9.4 s	Max Potential First Input Delay	230 ms

View Trace

Values are estimated and may vary. The performance score is based only on these metrics.

性能指标

观察上图，我们可以发现性能指标下面一共有六项内容，这六项内容分别对应了从Web应用的加载到页面展示完成的这段时间中，各个阶段所消耗的时长。在中间还有一个View Trace按钮，点击该按钮可以跳转到Performance标签，并且查看这些阶段在Performance中所对应的位置。最下方是加载过程中各个时间段的屏幕截图。

报告的第二个部分是**可优化项(Opportunities)**，如下图所示：

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Eliminate render-blocking resources	0.76 s
Preconnect to required origins	0.38 s
Warnings: A preconnect <link> was found for "https://api.bilibili.com" but was not used by the browser. Check that you are using the `crossorigin` attribute properly.	
Properly size images	0.16 s

可优化项(Opportunities)

这些可优化项是Audits发现页面中的一些可以直接优化的部分，你可以对照Audits给的这些提示来优化你的Web应用。

报告的第三部分是**手动诊断(Diagnostics)**，如下图所示：

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- Avoid enormous network payloads — Total size was 4,257 KB
- Serve static assets with an efficient cache policy — 7 resources found
- Avoid an excessive DOM size — 892 elements
- Avoid chaining critical requests — 12 chains found
- Keep request counts low and transfer sizes small — 236 requests • 4,257 KB

Passed audits (16)

手动诊断(Diagnostics)

在手动诊断部分，采集了一些可能存在性能问题的指标，这些指标可能会影响到页面的加载性能，Audits把详情列出来，并让你依据实际情况，来手动排查每一项。

报告的最后一部分是**运行时设置(Runtime Settings)**，如下图所示：

Runtime Settings

URL	https://m.bilibili.com/index.html
Fetch time	Nov 30, 2019, 9:22 AM GMT+8
Device	Emulated Nexus 5X
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3980.0 Safari/537.36
User agent (network)	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Mobile Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1612

Generated by **Lighthouse** 5.7.0 | File an issue

运行时设置(Runtime Settings)

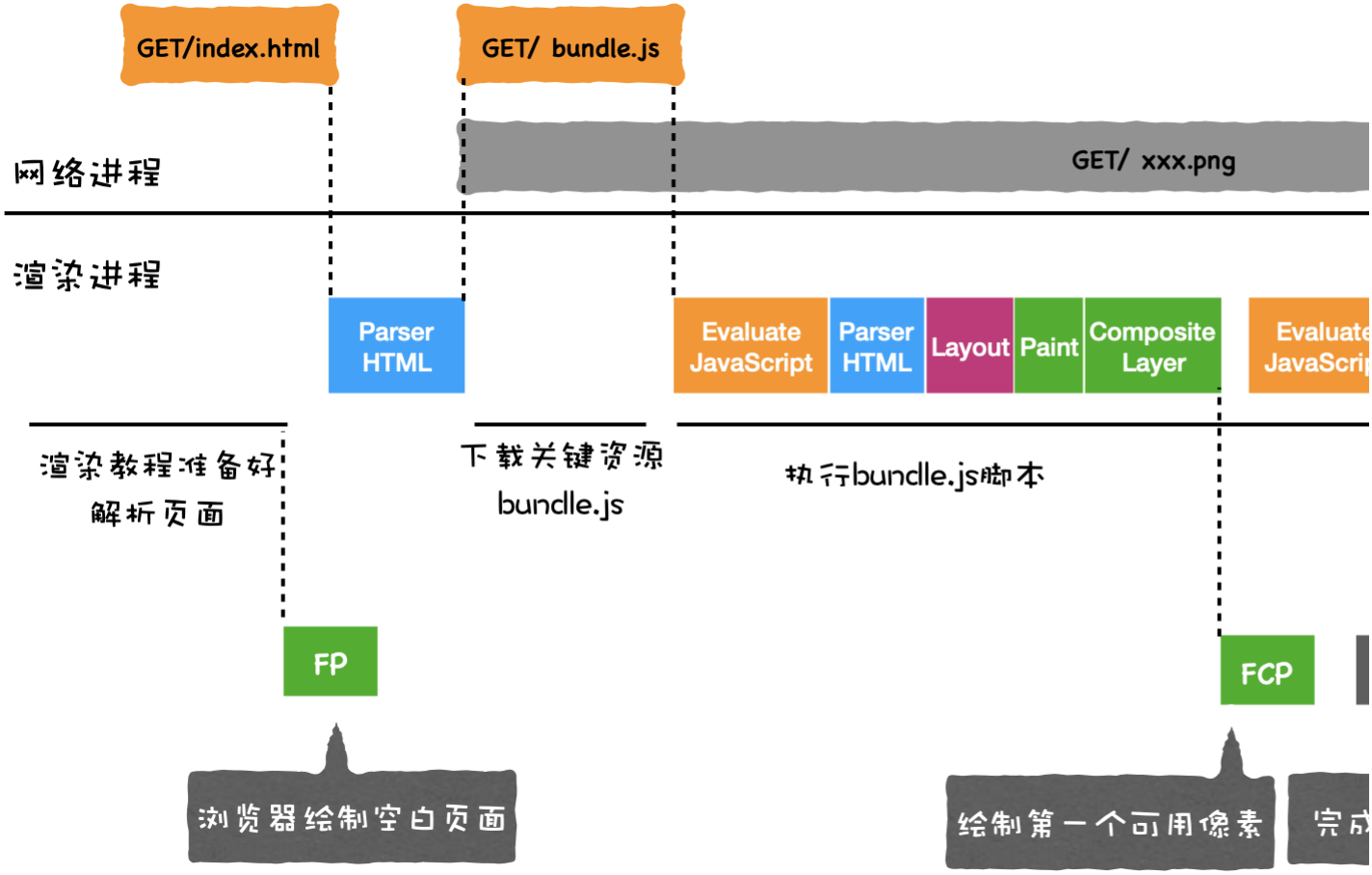
观察上图，这是运行时的一些基本数据，如果选择移动设备模式，你可以看到发送网络请求时的User Agent 会变成设备相关信息，还有会模拟设备的网速，这个体现在网络限速上。

根据性能报告优化Web性能

现在有了性能报告，接下来我们就可以依据报告来分析如何优化Web应用了。最直接的方式是想办法提高性能指标的分数，而性能指标的分数是由六项指标决定的，它们分别是：

1. 首次绘制(First Paint)；
2. 首次有效绘制(First Meaningfull Paint)；
3. 首屏时间(Speed Index)；
4. 首次CPU空闲时间(First CPU Idle)；
5. 完全可交互时间(Time to Interactive)；
6. 最大估计输入延时(Max Potential First Input Delay)。

那么接下来我会逐一分析六项指标的含义，并讨论如何提升这六项指标的数值。这六项都是页面在加载过程中的性能指标，所以要弄明白这六项指标的具体含义，我们还得结合页面的加载过程来分析。一图胜过千言，我们还是先看下面这张页面从加载到展示的过程图：



页面加载过程

观察上图的页面加载过程，我们发现，在渲染进程确认要渲染当前的请求后，渲染进程会创建一个空白页面，我们把创建空白页面的这个时间点称为**First Paint**，简称**FP**。

然后渲染进程继续请求关键资源，我们在《25 | 页面性能：如何系统地优化页面？》这节中介绍过了关键资源，并且知道了关键资源包括了JavaScript文件和CSS文件，因为关键资源会阻塞页面的渲染，所以我们需要等待关键资源加载完成后，才能执行进一步的页面绘制。

上图中，bundle.js是关键资源，因此需要完成加载之后，渲染进程才能执行该脚本，然后脚本会修改DOM，引发重绘和重排等一系列操作，当页面中绘制了第一个像素时，我们把这个时间点称为**First Content Paint**，简称**FCP**。

接下来继续执行JavaScript脚本，当首屏内容完全绘制完成时，我们把这个时间点称为**Largest Content Paint**，简称**LCP**。

在FCP和LCP中间，还有一个FMP，这个是首次有效绘制，由于FMP计算复杂，而且容易出错，现在不推荐使用该指标，所以这里我们也不做过多介绍了。

接下来JavaScript脚本执行结束，渲染进程判断该页面的DOM生成完毕，于是触发DOMContentLoad事件。等所有资源都加载结束之后，再触发onload事件。

好了，以上就是页面在加载过程中各个重要的时间节点，了解了这些时间节点，我们就可以来聊聊性能报告的六项指标的含义并讨论如何优化这些指标。

我们先来分析下**第一项指标FP**，如果FP时间过久，那么直接说明了一个问题，那就是页面的HTML文件可能由于网络原因导致加载时间过久，这块具体的分析过程你可以参考《21 | Chrome开发者工具：利用网络面板做性能分析》这节内容。

第二项是FMP，上面也提到过由于FMP计算复杂，所以现在不建议使用该指标了，另外由于LCP的计算规则简单，所以推荐使用LCP指标，具体文章你可以参考这里。不过是FMP还是LCP，优化它们的方式都是类似的，你可以结合上图，如果FMP和LCP消耗时间过久，那么有可能是加载关键资源花的时间过久，也有可能是JavaScript执行过程中所花的时间过久，所以我们可以针对具体的情况来具体分析。

第三项是首屏时间(Speed Index)，这就是我们上面提到的LCP，它表示填满首屏页面所消耗的时间，首屏时间的值越大，那么加载速度越慢，具体的优化方式同优化第二项FMP是一样。

第四项是首次CPU空闲时间(First CPU Idle)，也称为First Interactive，它表示页面达到最小化可交互的时间，也就是说并不需要等到页面上的所有元素都可交互，只要可以对大部分用户输入做出响应即可。要缩短首次CPU空闲时长，我们就需要尽可能快地加载完关键资源，尽可能快地渲染出来首屏内容，因此优化方式和第二项FMP和第三项LCP是一样的。

第五项是完全可交互时间(Time to Interactive)，简称TTI，它表示页面中所有元素都达到了可交互的时长。简单理解就这时候页面的内容已经完全显示出来了，所有的JavaScript事件已经注册完成，页面能够对用户的交互做出快速响应，通常满足响应速度在50毫秒以内。如果要解决TTI时间过久的问题，我们可以推迟执行一些和生成页面无关的JavaScript工作。

第六项是最大估计输入延时(Max Potential First Input Delay），这个指标是估计你的Web页面在加载最繁忙的阶段， 窗口中响应用户输入所需的时间，为了改善该指标，我们可以使用WebWorker来执行一些计算，从而释放主线程。另一个有用的措施是重构CSS选择器，以确保它们执行较少的计算。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容：

本文我们主要讨论如何优化加载阶段的Web应用的性能。

要想优化Web性能，首先得需要有Web应用的性能数据。所以接下来，我们介绍了Chrome采集Web性能数据的两个工具：Performance和Audits，Performance可以采集非常多的性能，但是其使用难度大，相反，Audtis就简单了许多，它会分析检测到的性能数据并给出站点的性能得分，同时，还会给我们提供一些优化建议。

我们先从简单的工具上手，所以本文我们主要分析了Audits的使用方式，先介绍了如何使用Audits生成性能报告，然后我们解读了性能报告中的每一项内容。

大致了解Audits生成的性能报告之后，我们又分析Web应用在加载阶段的几个关键时间点，最后我们分析性能指标的具体含义以及如何提高性能指标的分数，从而达到优化Web应用的目的。

通过介绍，我们知道了Audits非常适合用来分析加载阶段的Web性能，除此之外，Audits还有其他非常实用的功能，比如可以检测我们的代码是否符合一些最佳实践，并给出提示，这样我们就可以根据Audits的提示来决定是否需要优化我们的代码，这个功能非常不错，具体使用方式留给你自己去摸索了。

课后思考

在文中我们又分析Web应用在加载阶段的几个关键时间点，在Audits中，通过对这些时间点的分析，输出了文中介绍的六项性能指标，其实这些时间点也可以通过Performance的时间线(Timelines)来查看，那么今天留给你的任务是：提前熟悉下Performance工具，并对照这文中加载阶段的几个时间点来熟悉下Performance的时间线(Timelines)，欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

你好，我是李兵。

作为一名前端工程师，除了需要编写功能性的代码以外，我们还需要关注Web应用的性能问题，我们应该有能力让我们的Web应用占用最小的资源，并以最高性能运行，这也是前端工程师进阶的必要能力。既然性能这么重要，那么我们今天要来聊聊Web性能问题。

到底什么是Web性能？

我们看下wiki对Web 性能的定义：

> Web 性能描述了Web应用在浏览器上的加载和显示的速度。

因此，当我们讨论Web性能时，其实就是讨论Web应用速度，关于Web应用的速度，我们需要从两个阶段来考虑：

- 页面加载阶段；
- 页面交互阶段。

在本文中，我们会将焦点放到第一个阶段：页面加载阶段的性能，在下篇文章中，我们会来重点分析页面交互阶段的性能。

性能检测工具：Performance vs Audits

要想优化Web的性能，我们就需要有监控Web应用的性能数据，那怎么监控呢？

如果没有工具来模拟各种不同的场景并统计各种性能指标，那么定位Web应用的性能瓶颈将是一件非常困难的任务。幸好，Chrome为我们提供了非常完善的性能检测工具：**Performance**和**Audits**，它们能够准确统计页面在加载阶段和运行阶段的一些核心数据，诸如任务执行记录、首屏展示花费的时长等，有了这些数据我们就能很容易定位到Web应用的**性能瓶颈** 。

首先Performance非常强大，因为它为我们提供了非常多的运行时数据，利用这些数据我们就可以分析出来Web应用的瓶颈。但是要完全学会其使用方式却是非常有难度的，其难点在于这些数据涉及到了特别多的概念，而这些概念又和浏览器的系统架构、消息循环机制、渲染流水线等知识紧密联系在了一起。

相反，Audtis就简单了许多，它将检测到的细节数据隐藏在背后，只提供给我们一些直观的性能数据，同时，还会给我们提供一些优化建议。

Perfomance能让我们看到更多细节数据，但是更加复杂，Audits就比较智能，但是隐藏了更多细节。为了能够让你循序渐进地理解内容，所以本节我们先从简单的Audits入手，看看如何利用它来检测和优化页面在加载阶段的性能，然后在下一节我们再来分析Perfomance。

检测之前准备工作

不过在检测Web的性能指标之前，我们还要配置好工作环境，具体地讲，你需要准备以下内容：

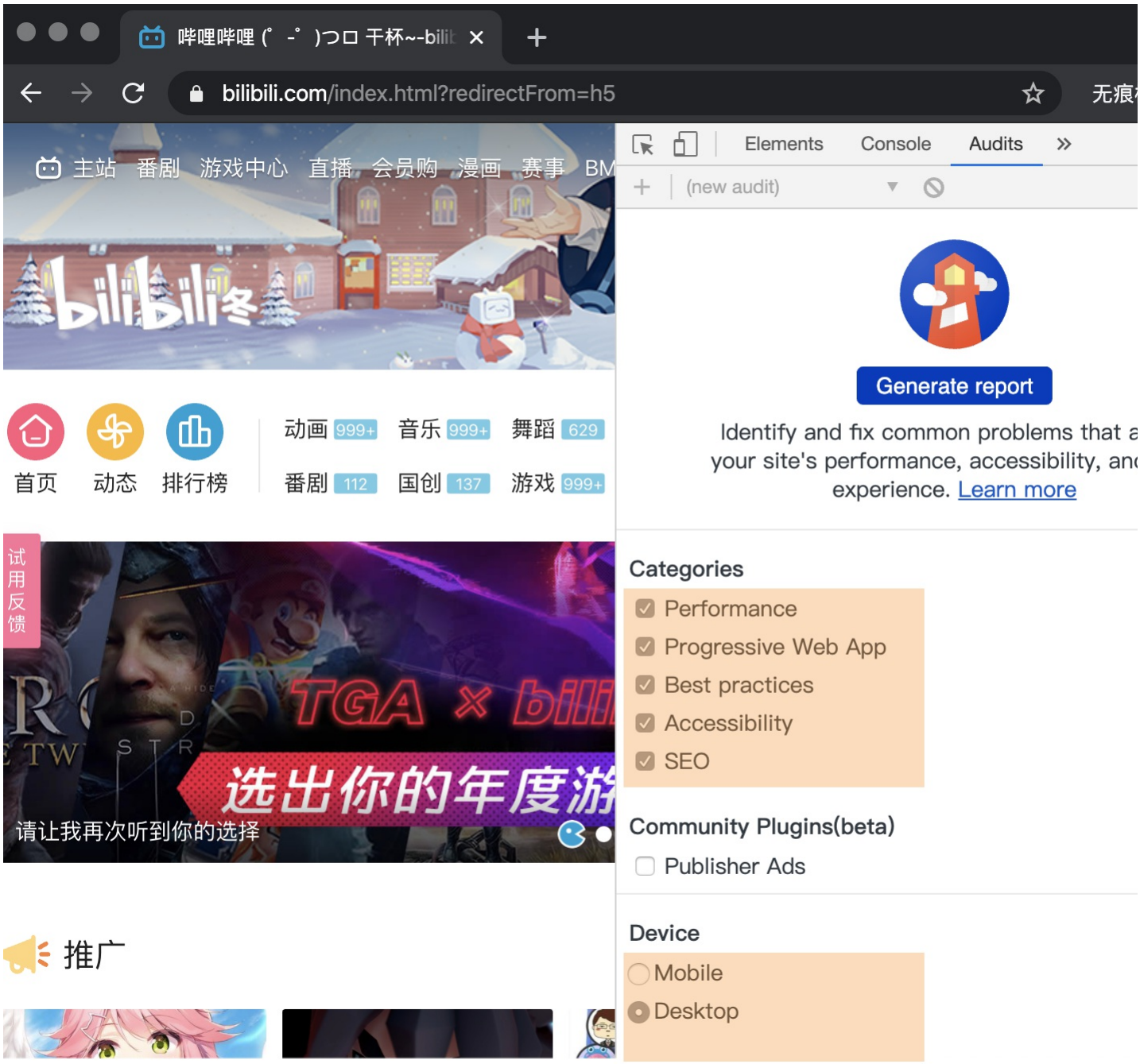
- 首先准备Chrome Canary版的浏览器，Chrome Canary是采用最新技术构建的，它的开发者工具和浏览器特性都是最新的，所以我推荐你使用Chrome Canary来做性能分析。当然你也可以使用稳定版的Chrome。
- 然后我们需要在Chrome的隐身模式下工作，这样可以确保我们安装的扩展、浏览器缓存、Cookie等数据不会影响到检测结果。

利用Audits生成Web性能报告

环境准备好了之后，我们就可以生成站点在加载阶段的性能报告了，这里我们可以拿B站作为分析的列子。

- 首先我们打开浏览器的隐身窗口，Windows系统下面的快捷键是Control+Shift+N，Mac系统下面的快捷键是Command+Shift+N。
- 然后在隐身窗口中输入B站的网站。
- 打开Chrome的开发者工具，选择Audits标签。

最终打开的页面如下图所示：



Audits界面

观察上图中的Audits界面，我们可以看到，在生成报告之前，我们需要先配置Audits，配置模块主要有两部分组成，一个是**监测类型(Categories)**，另外一个是**设备类型(Device)**。

监控类型(Categories)是指需要监控哪些内容，这里有五个对应的选项，它们的功能分别是：

- 监测并分析Web性能(**Performance**)；
- 监测并分析PWA(**Progressive Web App**)程序的性能；
- 监测并分析Web应用是否采用了最佳实践策略(**Best practices**)；
- 监测并分析是否实施了无障碍功能(**Accessibility**)，无障碍功能让一些身体有障碍的人可以方便地浏览你的Web应用。
- 监测并分析Web应用是否采实施了SEO搜索引擎优化(**SEO**)。

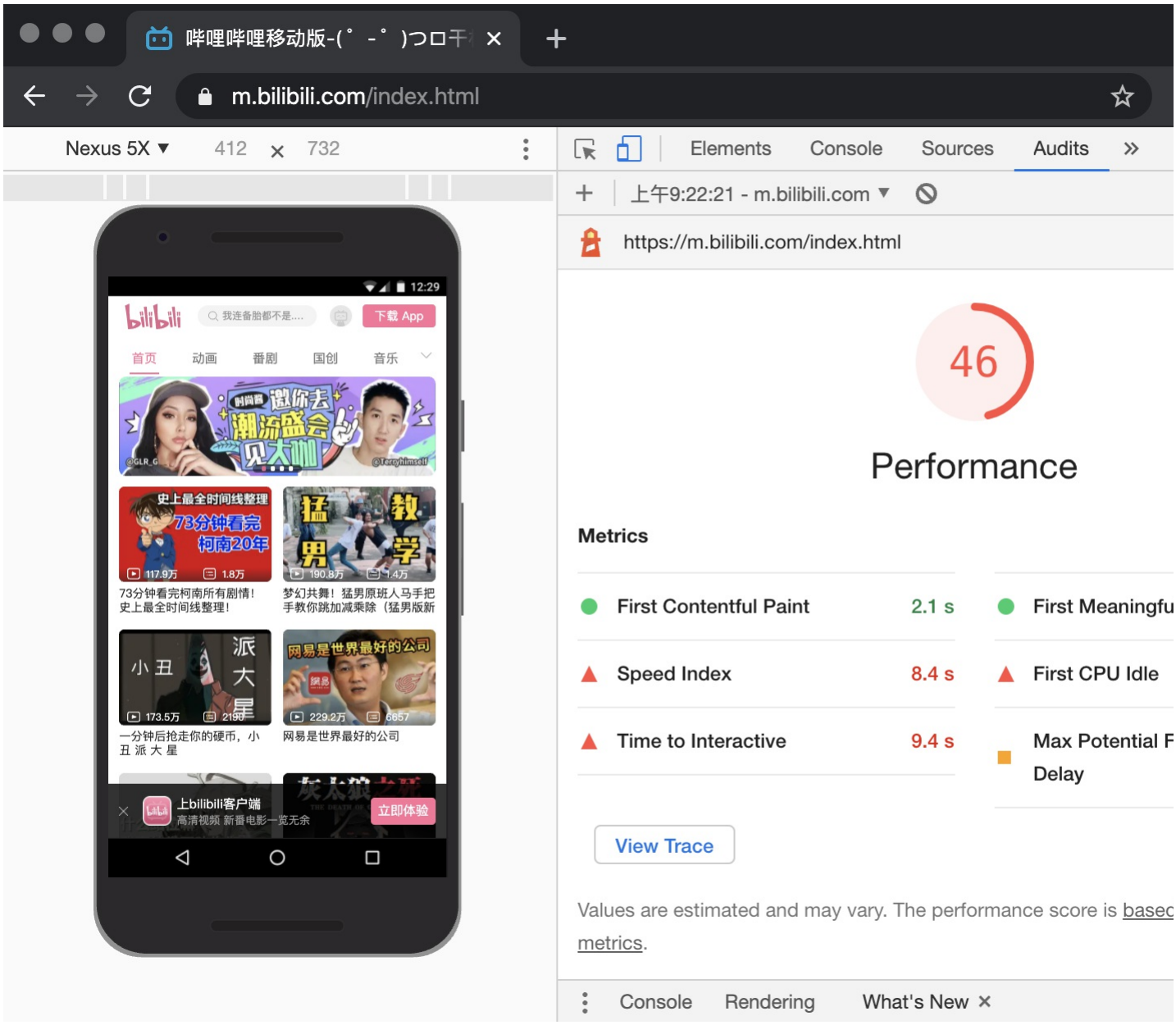
本文我们只需要关注Web应用的加载性能，所以勾选第一个Performance选项就可以了。

再看看**设备(Device)部分**，它给了我们两个选项，Moblie选项是用来模拟移动设备环境的，另外一个Desktop选项是用来模拟桌面环境的。这里我们选择移动设备选项，因为目前大多数流量都是由移动设备产生的，所以移动设备上的Web性能显得更加重要。

配置好选项之后，我们就可以点击最上面的生成报告(Generate report)按钮来生成报告了。

解读性能报告

点击生成报告的按钮之后，我们大约需要等待一分钟左右，Audits就可以生成最终的分析报告了，如下图所示：



生成的报告图

观察上图的分析报告，中间圆圈中的数字表示该站点在加载过程中的总体Web性能得分，总分是100分。我们目前的得分为46分，这表示该站点加载阶段的性能还有很大的提升空间。

Audits除了生成性能指标以外，还会分析该站点并提供了很多优化建议，我们可以根据这些建议来改进Web应用以获得更高的得分，进而获得更好的用户体验效果。

既能分析Web性能得分又能给出优化建议，所以Audits的分析报告还是非常有价值的，那么接下来，我们就来解读下Audits生成的性能报告。

报告的第一个部分是**性能指标(Metrics)**，如下图所示：

Metrics

First Contentful Paint	2.1 s	First Meaningful Paint	2.1 s
Speed Index	8.4 s	First CPU Idle	7.2 s
Time to Interactive	9.4 s	Max Potential First Input Delay	230 ms

View Trace

Values are estimated and may vary. The performance score is based only on these metrics.

性能指标

观察上图，我们可以发现性能指标下面一共有六项内容，这六项内容分别对应了从Web应用的加载到页面展示完成的这段时间中，各个阶段所消耗的时长。在中间还有一个View Trace按钮，点击该按钮可以跳转到Performance标签，并且查看这些阶段在Performance中所对应的位置。最下方是加载过程中各个时间段的屏幕截图。

报告的第二个部分是**可优化项(Opportunities)**，如下图所示：

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Eliminate render-blocking resources	0.76 s
Preconnect to required origins	0.38 s
Warnings: A preconnect <link> was found for "https://api.bilibili.com" but was not used by the browser. Check that you are using the `crossorigin` attribute properly.	
Properly size images	0.16 s

可优化项(Opportunities)

这些可优化项是Audits发现页面中的一些可以直接优化的部分，你可以对照Audits给的这些提示来优化你的Web应用。

报告的第三部分是**手动诊断(Diagnostics)**，如下图所示：

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- Avoid enormous network payloads — Total size was 4,257 KB
- Serve static assets with an efficient cache policy — 7 resources found
- Avoid an excessive DOM size — 892 elements
- Avoid chaining critical requests — 12 chains found
- Keep request counts low and transfer sizes small — 236 requests • 4,257 KB

Passed audits (16)

手动诊断(Diagnostics)

在手动诊断部分，采集了一些可能存在性能问题的指标，这些指标可能会影响到页面的加载性能，Audits把详情列出来，并让你依据实际情况，来手动排查每一项。

报告的最后一部分是**运行时设置(Runtime Settings)**，如下图所示：

Runtime Settings

URL	https://m.bilibili.com/index.html
Fetch time	Nov 30, 2019, 9:22 AM GMT+8
Device	Emulated Nexus 5X
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3980.0 Safari/537.36
User agent (network)	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Mobile Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1612

Generated by **Lighthouse** 5.7.0 | File an issue

运行时设置(Runtime Settings)

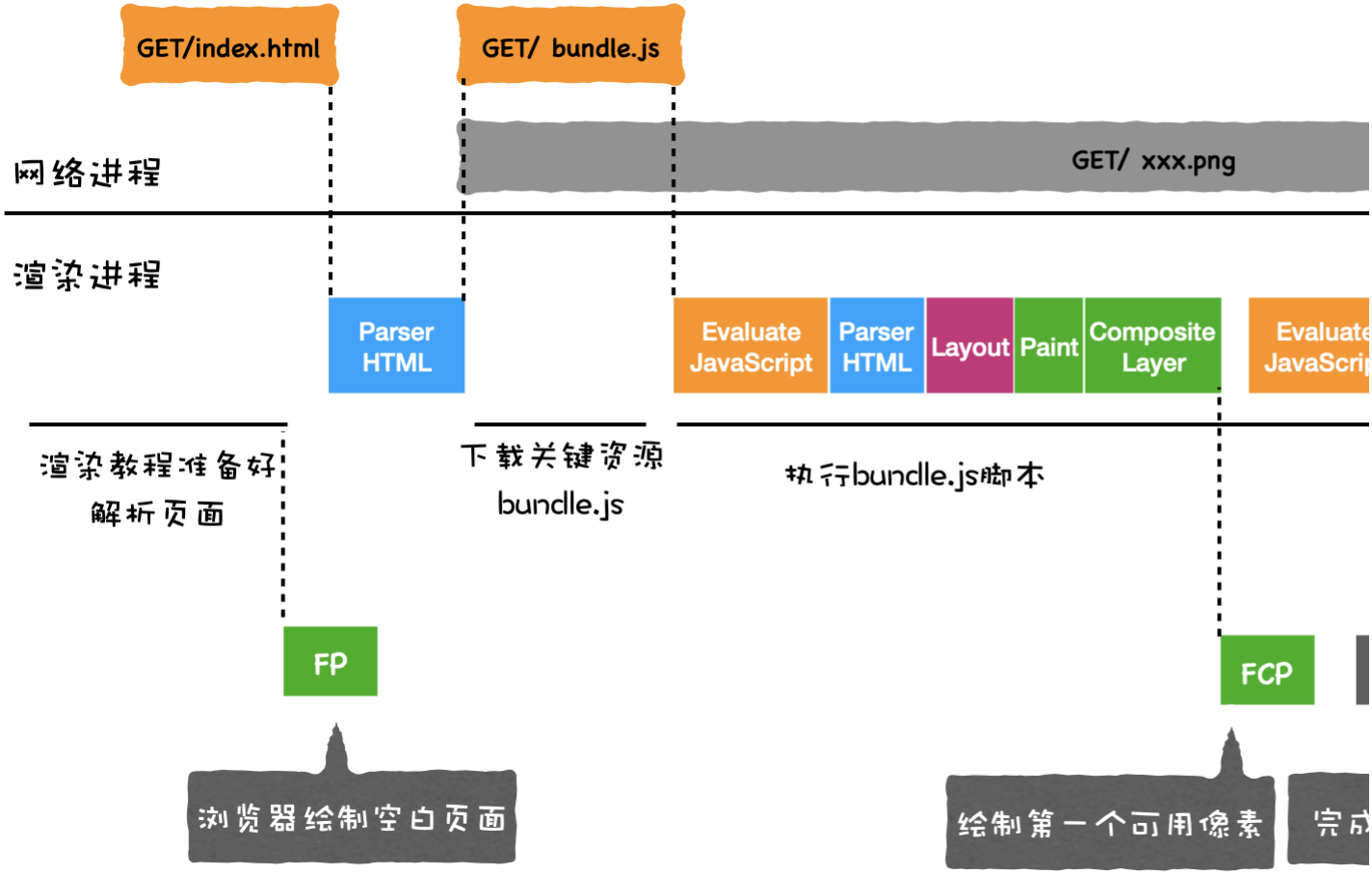
观察上图，这是运行时的一些基本数据，如果选择移动设备模式，你可以看到发送网络请求时的User Agent 会变成设备相关信息，还有会模拟设备的网速，这个体现在网络限速上。

根据性能报告优化Web性能

现在有了性能报告，接下来我们就可以依据报告来分析如何优化Web应用了。最直接的方式是想办法提高性能指标的分数，而性能指标的分数是由六项指标决定的，它们分别是：

1. 首次绘制(First Paint)；
2. 首次有效绘制(First Meaningfull Paint)；
3. 首屏时间(Speed Index)；
4. 首次CPU空闲时间(First CPU Idle)；
5. 完全可交互时间(Time to Interactive)；
6. 最大估计输入延时(Max Potential First Input Delay)。

那么接下来我会逐一分析六项指标的含义，并讨论如何提升这六项指标的数值。这六项都是页面在加载过程中的性能指标，所以要弄明白这六项指标的具体含义，我们还得结合页面的加载过程来分析。一图胜过千言，我们还是先看下面这张页面从加载到展示的过程图：



页面加载过程

观察上图的页面加载过程，我们发现，在渲染进程确认要渲染当前的请求后，渲染进程会创建一个空白页面，我们把创建空白页面的这个时间点称为**First Paint**，简称**FP**。

然后渲染进程继续请求关键资源，我们在《25 | 页面性能：如何系统地优化页面？》这节中介绍过了关键资源，并且知道了关键资源包括了JavaScript文件和CSS文件，因为关键资源会阻塞页面的渲染，所以我们需要等待关键资源加载完成后，才能执行进一步的页面绘制。

上图中，bundle.js是关键资源，因此需要完成加载之后，渲染进程才能执行该脚本，然后脚本会修改DOM，引发重绘和重排等一系列操作，当页面中绘制了第一个像素时，我们把这个时间点称为**First Content Paint**，简称**FCP**。

接下来继续执行JavaScript脚本，当首屏内容完全绘制完成时，我们把这个时间点称为**Largest Content Paint**，简称**LCP**。

在FCP和LCP中间，还有一个FMP，这个是首次有效绘制，由于FMP计算复杂，而且容易出错，现在不推荐使用该指标，所以这里我们也不做过多介绍了。

接下来JavaScript脚本执行结束，渲染进程判断该页面的DOM生成完毕，于是触发DOMContentLoad事件。等所有资源都加载结束之后，再触发onload事件。

好了，以上就是页面在加载过程中各个重要的时间节点，了解了这些时间节点，我们就可以来聊聊性能报告的六项指标的含义并讨论如何优化这些指标。

我们先来分析下**第一项指标FP**，如果FP时间过久，那么直接说明了一个问题，那就是页面的HTML文件可能由于网络原因导致加载时间过久，这块具体的分析过程你可以参考《21 | Chrome开发者工具：利用网络面板做性能分析》这节内容。

第二项是FMP，上面也提到过由于FMP计算复杂，所以现在不建议使用该指标了，另外由于LCP的计算规则简单，所以推荐使用LCP指标，具体文章你可以参考这里。不过是FMP还是LCP，优化它们的方式都是类似的，你可以结合上图，如果FMP和LCP消耗时间过久，那么有可能是加载关键资源花的时间过久，也有可能是JavaScript执行过程中所花的时间过久，所以我们可以针对具体的情况来具体分析。

第三项是首屏时间(Speed Index)，这就是我们上面提到的LCP，它表示填满首屏页面所消耗的时间，首屏时间的值越大，那么加载速度越慢，具体的优化方式同优化第二项FMP是一样。

第四项是首次CPU空闲时间(First CPU Idle)，也称为First Interactive，它表示页面达到最小化可交互的时间，也就是说并不需要等到页面上的所有元素都可交互，只要可以对大部分用户输入做出响应即可。要缩短首次CPU空闲时长，我们就需要尽可能快地加载完关键资源，尽可能快地渲染出来首屏内容，因此优化方式和第二项FMP和第三项LCP是一样的。

第五项是完全可交互时间(Time to Interactive)，简称TTI，它表示页面中所有元素都达到了可交互的时长。简单理解就这时候页面的内容已经完全显示出来了，所有的JavaScript事件已经注册完成，页面能够对用户的交互做出快速响应，通常满足响应速度在50毫秒以内。如果要解决TTI时间过久的问题，我们可以推迟执行一些和生成页面无关的JavaScript工作。

第六项是最大估计输入延时(Max Potential First Input Delay），这个指标是估计你的Web页面在加载最繁忙的阶段， 窗口中响应用户输入所需的时间，为了改善该指标，我们可以使用WebWorker来执行一些计算，从而释放主线程。另一个有用的措施是重构CSS选择器，以确保它们执行较少的计算。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容：

本文我们主要讨论如何优化加载阶段的Web应用的性能。

要想优化Web性能，首先得需要有Web应用的性能数据。所以接下来，我们介绍了Chrome采集Web性能数据的两个工具：Performance和Audits，Performance可以采集非常多的性能，但是其使用难度大，相反，Audtis就简单了许多，它会分析检测到的性能数据并给出站点的性能得分，同时，还会给我们提供一些优化建议。

我们先从简单的工具上手，所以本文我们主要分析了Audits的使用方式，先介绍了如何使用Audits生成性能报告，然后我们解读了性能报告中的每一项内容。

大致了解Audits生成的性能报告之后，我们又分析Web应用在加载阶段的几个关键时间点，最后我们分析性能指标的具体含义以及如何提高性能指标的分数，从而达到优化Web应用的目的。

通过介绍，我们知道了Audits非常适合用来分析加载阶段的Web性能，除此之外，Audits还有其他非常实用的功能，比如可以检测我们的代码是否符合一些最佳实践，并给出提示，这样我们就可以根据Audits的提示来决定是否需要优化我们的代码，这个功能非常不错，具体使用方式留给你自己去摸索了。

课后思考

在文中我们又分析Web应用在加载阶段的几个关键时间点，在Audits中，通过对这些时间点的分析，输出了文中介绍的六项性能指标，其实这些时间点也可以通过Performance的时间线(Timelines)来查看，那么今天留给你的任务是：提前熟悉下Performance工具，并对照这文中加载阶段的几个时间点来熟悉下Performance的时间线(Timelines)，欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

你好，我是李兵。

作为一名前端工程师，除了需要编写功能性的代码以外，我们还需要关注Web应用的性能问题，我们应该有能力让我们的Web应用占用最小的资源，并以最高性能运行，这也是前端工程师进阶的必要能力。既然性能这么重要，那么我们今天要来聊聊Web性能问题。

到底什么是Web性能？

我们看下wiki对Web 性能的定义：

> Web 性能描述了Web应用在浏览器上的加载和显示的速度。

因此，当我们讨论Web性能时，其实就是讨论Web应用速度，关于Web应用的速度，我们需要从两个阶段来考虑：

- 页面加载阶段；
- 页面交互阶段。

在本文中，我们会将焦点放到第一个阶段：页面加载阶段的性能，在下篇文章中，我们会来重点分析页面交互阶段的性能。

性能检测工具：Performance vs Audits

要想优化Web的性能，我们就需要有监控Web应用的性能数据，那怎么监控呢？

如果没有工具来模拟各种不同的场景并统计各种性能指标，那么定位Web应用的性能瓶颈将是一件非常困难的任务。幸好，Chrome为我们提供了非常完善的性能检测工具：**Performance**和**Audits**，它们能够准确统计页面在加载阶段和运行阶段的一些核心数据，诸如任务执行记录、首屏展示花费的时长等，有了这些数据我们就能很容易定位到Web应用的**性能瓶颈** 。

首先Performance非常强大，因为它为我们提供了非常多的运行时数据，利用这些数据我们就可以分析出来Web应用的瓶颈。但是要完全学会其使用方式却是非常有难度的，其难点在于这些数据涉及到了特别多的概念，而这些概念又和浏览器的系统架构、消息循环机制、渲染流水线等知识紧密联系在了一起。

相反，Audtis就简单了许多，它将检测到的细节数据隐藏在背后，只提供给我们一些直观的性能数据，同时，还会给我们提供一些优化建议。

Perfomance能让我们看到更多细节数据，但是更加复杂，Audits就比较智能，但是隐藏了更多细节。为了能够让你循序渐进地理解内容，所以本节我们先从简单的Audits入手，看看如何利用它来检测和优化页面在加载阶段的性能，然后在下一节我们再来分析Perfomance。

检测之前准备工作

不过在检测Web的性能指标之前，我们还要配置好工作环境，具体地讲，你需要准备以下内容：

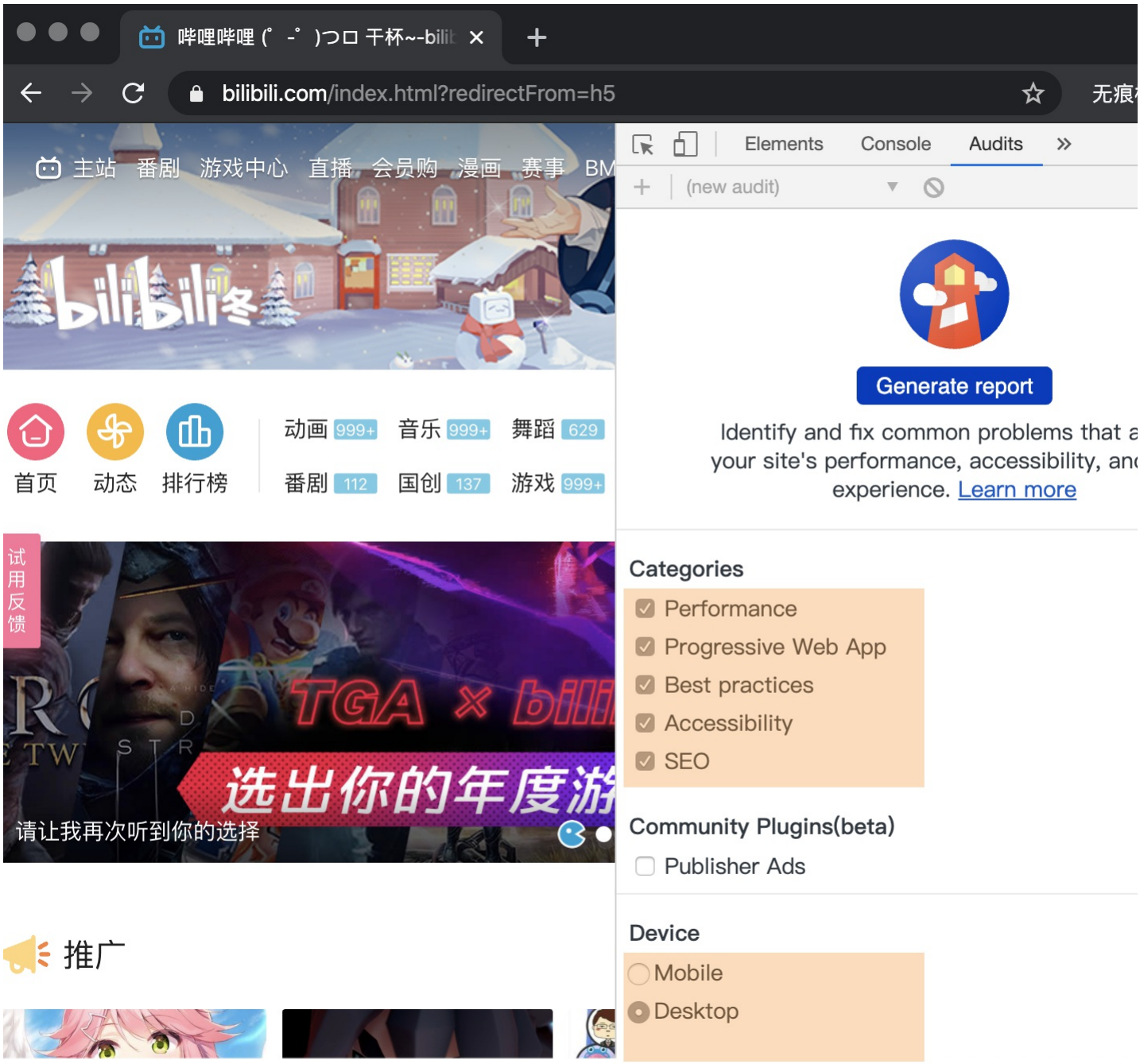
- 首先准备Chrome Canary版的浏览器，Chrome Canary是采用最新技术构建的，它的开发者工具和浏览器特性都是最新的，所以我推荐你使用Chrome Canary来做性能分析。当然你也可以使用稳定版的Chrome。
- 然后我们需要在Chrome的隐身模式下工作，这样可以确保我们安装的扩展、浏览器缓存、Cookie等数据不会影响到检测结果。

利用Audits生成Web性能报告

环境准备好了之后，我们就可以生成站点在加载阶段的性能报告了，这里我们可以拿B站作为分析的列子。

- 首先我们打开浏览器的隐身窗口，Windows系统下面的快捷键是Control+Shift+N，Mac系统下面的快捷键是Command+Shift+N。
- 然后在隐身窗口中输入B站的网站。
- 打开Chrome的开发者工具，选择Audits标签。

最终打开的页面如下图所示：



Audits界面

观察上图中的Audits界面，我们可以看到，在生成报告之前，我们需要先配置Audits，配置模块主要有两部分组成，一个是**监测类型(Categories)**，另外一个是**设备类型(Device)**。

监控类型(Categories)是指需要监控哪些内容，这里有五个对应的选项，它们的功能分别是：

- 监测并分析Web性能(**Performance**)；
- 监测并分析PWA(**Progressive Web App**)程序的性能；
- 监测并分析Web应用是否采用了最佳实践策略(**Best practices**)；
- 监测并分析是否实施了无障碍功能(**Accessibility**)，无障碍功能让一些身体有障碍的人可以方便地浏览你的Web应用。
- 监测并分析Web应用是否采实施了SEO搜索引擎优化(**SEO**)。

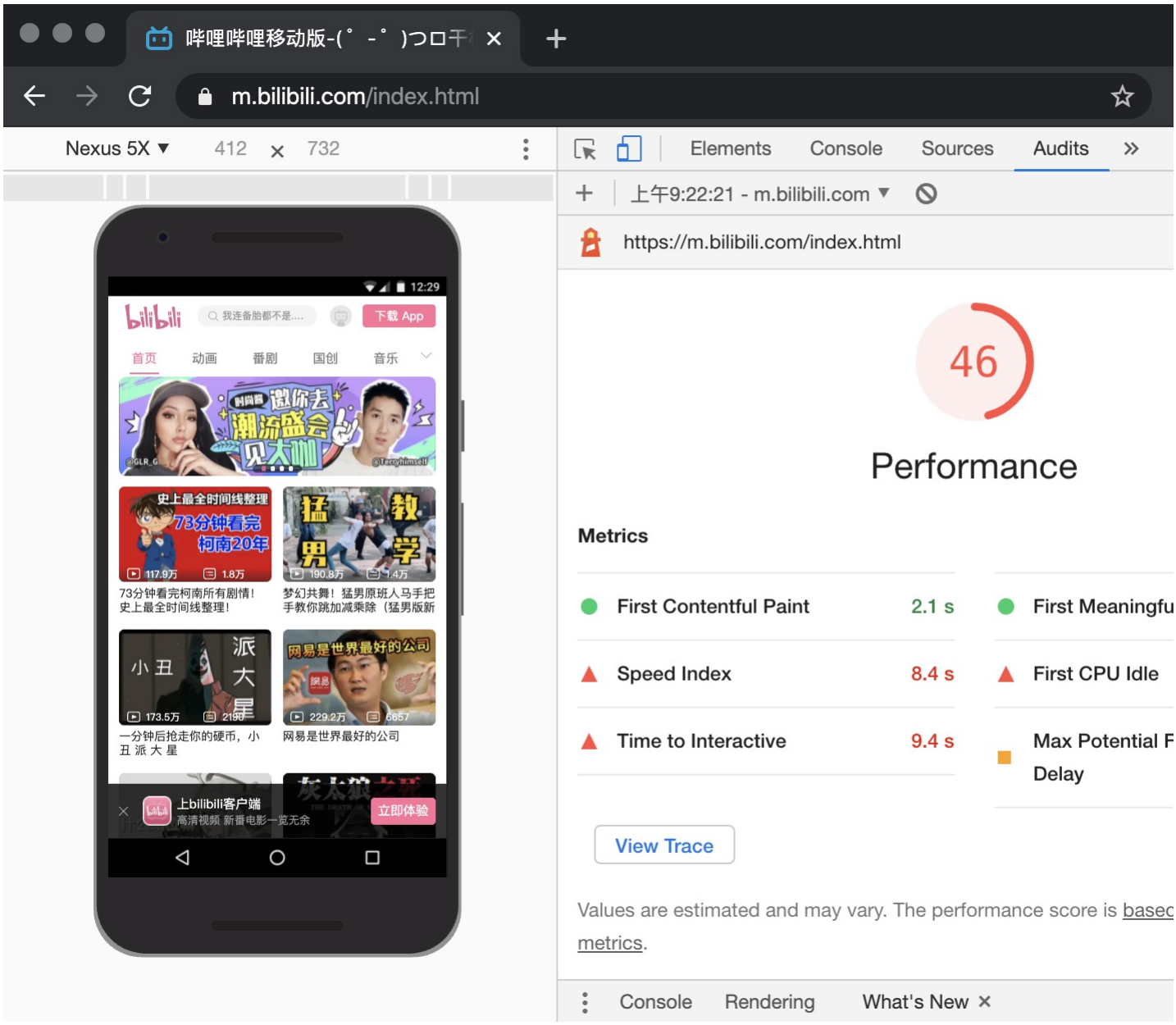
本文我们只需要关注Web应用的加载性能，所以勾选第一个Performance选项就可以了。

再看看**设备(Device)部分**，它给了我们两个选项，Moblie选项是用来模拟移动设备环境的，另外一个Desktop选项是用来模拟桌面环境的。这里我们选择移动设备选项，因为目前大多数流量都是由移动设备产生的，所以移动设备上的Web性能显得更加重要。

配置好选项之后，我们就可以点击最上面的生成报告(Generate report)按钮来生成报告了。

解读性能报告

点击生成报告的按钮之后，我们大约需要等待一分钟左右，Audits就可以生成最终的分析报告了，如下图所示：



生成的报告图

观察上图的分析报告，中间圆圈中的数字表示该站点在加载过程中的总体Web性能得分，总分是100分。我们目前的得分为46分，这表示该站点加载阶段的性能还有很大的提升空间。

Audits除了生成性能指标以外，还会分析该站点并提供了很多优化建议，我们可以根据这些建议来改进Web应用以获得更高的得分，进而获得更好的用户体验效果。

既能分析Web性能得分又能给出优化建议，所以Audits的分析报告还是非常有价值的，那么接下来，我们就来解读下Audits生成的性能报告。

报告的第一个部分是**性能指标(Metrics)**，如下图所示：

Metrics

First Contentful Paint	2.1 s	First Meaningful Paint	2.1 s
Speed Index	8.4 s	First CPU Idle	7.2 s
Time to Interactive	9.4 s	Max Potential First Input Delay	230 ms

View Trace

Values are estimated and may vary. The performance score is based only on these metrics.

性能指标

观察上图，我们可以发现性能指标下面一共有六项内容，这六项内容分别对应了从Web应用的加载到页面展示完成的这段时间中，各个阶段所消耗的时长。在中间还有一个View Trace按钮，点击该按钮可以跳转到Performance标签，并且查看这些阶段在Performance中所对应的位置。最下方是加载过程中各个时间段的屏幕截图。

报告的第二个部分是**可优化项(Opportunities)**，如下图所示：

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Eliminate render-blocking resources	0.76 s
Preconnect to required origins	0.38 s
Warnings: A preconnect <link> was found for "https://api.bilibili.com" but was not used by the browser. Check that you are using the `crossorigin` attribute properly.	
Properly size images	0.16 s

可优化项(Opportunities)

这些可优化项是Audits发现页面中的一些可以直接优化的部分，你可以对照Audits给的这些提示来优化你的Web应用。

报告的第三部分是**手动诊断(Diagnostics)**，如下图所示：

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- Avoid enormous network payloads — Total size was 4,257 KB
- Serve static assets with an efficient cache policy — 7 resources found
- Avoid an excessive DOM size — 892 elements
- Avoid chaining critical requests — 12 chains found
- Keep request counts low and transfer sizes small — 236 requests • 4,257 KB

Passed audits (16)

手动诊断(Diagnostics)

在手动诊断部分，采集了一些可能存在性能问题的指标，这些指标可能会影响到页面的加载性能，Audits把详情列出来，并让你依据实际情况，来手动排查每一项。

报告的最后一部分是**运行时设置(Runtime Settings)**，如下图所示：

Runtime Settings

URL	https://m.bilibili.com/index.html
Fetch time	Nov 30, 2019, 9:22 AM GMT+8
Device	Emulated Nexus 5X
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3980.0 Safari/537.36
User agent (network)	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Mobile Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1612

Generated by **Lighthouse** 5.7.0 | File an issue

运行时设置(Runtime Settings)

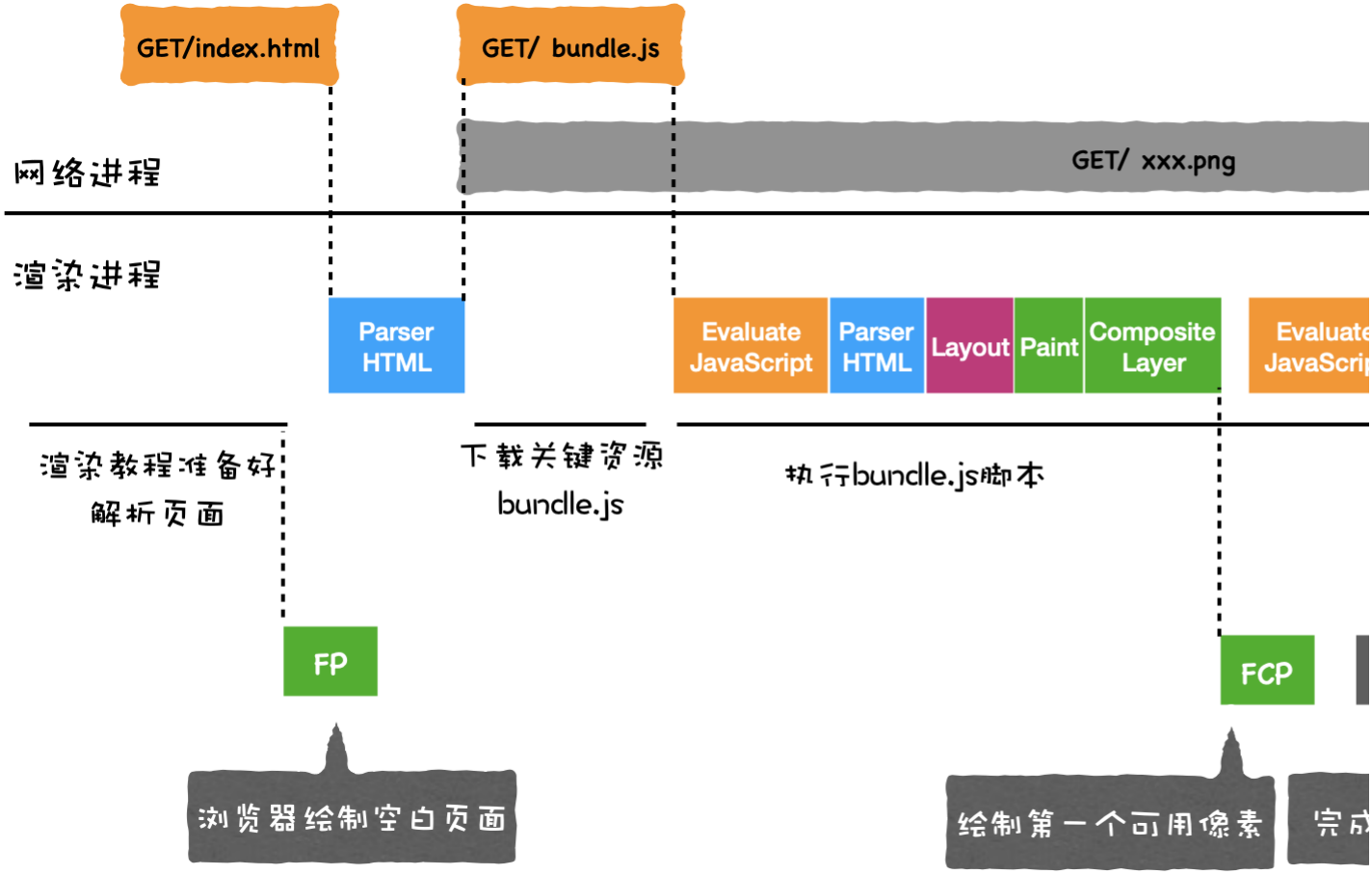
观察上图，这是运行时的一些基本数据，如果选择移动设备模式，你可以看到发送网络请求时的User Agent 会变成设备相关信息，还有会模拟设备的网速，这个体现在网络限速上。

根据性能报告优化Web性能

现在有了性能报告，接下来我们就可以依据报告来分析如何优化Web应用了。最直接的方式是想办法提高性能指标的分数，而性能指标的分数是由六项指标决定的，它们分别是：

1. 首次绘制(First Paint)；
2. 首次有效绘制(First Meaningfull Paint)；
3. 首屏时间(Speed Index)；
4. 首次CPU空闲时间(First CPU Idle)；
5. 完全可交互时间(Time to Interactive)；
6. 最大估计输入延时(Max Potential First Input Delay)。

那么接下来我会逐一分析六项指标的含义，并讨论如何提升这六项指标的数值。这六项都是页面在加载过程中的性能指标，所以要弄明白这六项指标的具体含义，我们还得结合页面的加载过程来分析。一图胜过千言，我们还是先看下面这张页面从加载到展示的过程图：



页面加载过程

观察上图的页面加载过程，我们发现，在渲染进程确认要渲染当前的请求后，渲染进程会创建一个空白页面，我们把创建空白页面的这个时间点称为**First Paint**，简称**FP**。

然后渲染进程继续请求关键资源，我们在《25丨页面性能：如何系统地优化页面？》这节中介绍过了关键资源，并且知道了关键资源包括了JavaScript文件和CSS文件，因为关键资源会阻塞页面的渲染，所以我们需要等待关键资源加载完成后，才能执行进一步的页面绘制。

上图中，bundle.js是关键资源，因此需要完成加载之后，渲染进程才能执行该脚本，然后脚本会修改DOM，引发重绘和重排等一系列操作，当页面中绘制了第一个像素时，我们把这个时间点称为**First Content Paint**，简称**FCP**。

接下来继续执行JavaScript脚本，当首屏内容完全绘制完成时，我们把这个时间点称为**Largest Content Paint**，简称**LCP**。

在FCP和LCP中间，还有一个FMP，这个是首次有效绘制，由于FMP计算复杂，而且容易出错，现在不推荐使用该指标，所以这里我们也不做过多介绍了。

接下来JavaScript脚本执行结束，渲染进程判断该页面的DOM生成完毕，于是触发DOMContentLoad事件。等所有资源都加载结束之后，再触发onload事件。

好了，以上就是页面在加载过程中各个重要的时间节点，了解了这些时间节点，我们就可以来聊聊性能报告的六项指标的含义并讨论如何优化这些指标。

我们先来分析下**第一项指标FP**，如果FP时间过久，那么直接说明了一个问题，那就是页面的HTML文件可能由于网络原因导致加载时间过久，这块具体的分析过程你可以参考《21丨Chrome开发者工具：利用网络面板做性能分析》这节内容。

第二项是FMP，上面也提到过由于FMP计算复杂，所以现在不建议使用该指标了，另外由于LCP的计算规则简单，所以推荐使用LCP指标，具体文章你可以参考这里。不过是FMP还是LCP，优化它们的方式都是类似的，你可以结合上图，如果FMP和LCP消耗时间过久，那么有可能是加载关键资源花的时间过久，也有可能是JavaScript执行过程中所花的时间过久，所以我们可以针对具体的情况来具体分析。

第三项是首屏时间(Speed Index)，这就是我们上面提到的LCP，它表示填满首屏页面所消耗的时间，首屏时间的值越大，那么加载速度越慢，具体的优化方式同优化第二项FMP是一样。

第四项是首次CPU空闲时间(First CPU Idle)，也称为First Interactive，它表示页面达到最小化可交互的时间，也就是说并不需要等到页面上的所有元素都可交互，只要可以对大部分用户输入做出响应即可。要缩短首次CPU空闲时长，我们就需要尽可能快地加载完关键资源，尽可能快地渲染出来首屏内容，因此优化方式和第二项FMP和第三项LCP是一样的。

第五项是完全可交互时间(Time to Interactive)，简称TTI，它表示页面中所有元素都达到了可交互的时长。简单理解就这时候页面的内容已经完全显示出来了，所有的JavaScript事件已经注册完成，页面能够对用户的交互做出快速响应，通常满足响应速度在50毫秒以内。如果要解决TTI时间过久的问题，我们可以推迟执行一些和生成页面无关的JavaScript工作。

第六项是最大估计输入延时(Max Potential First Input Delay)，这个指标是估计你的Web页面在加载最繁忙的阶段， 窗口中响应用户输入所需的时间，为了改善该指标，我们可以使用WebWorker来执行一些计算，从而释放主线程。另一个有用的措施是重构CSS选择器，以确保它们执行较少的计算。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容：

本文我们主要讨论如何优化加载阶段的Web应用的性能。

要想优化Web性能，首先得需要有Web应用的性能数据。所以接下来，我们介绍了Chrome采集Web性能数据的两个工具：Performance和Audits，Performance可以采集非常多的性能，但是其使用难度大，相反，Audtis就简单了许多，它会分析检测到的性能数据并给出站点的性能得分，同时，还会给我们提供一些优化建议。

我们先从简单的工具上手，所以本文我们主要分析了Audits的使用方式，先介绍了如何使用Audits生成性能报告，然后我们解读了性能报告中的每一项内容。

大致了解Audits生成的性能报告之后，我们又分析Web应用在加载阶段的几个关键时间点，最后我们分析性能指标的具体含义以及如何提高性能指标的分数，从而达到优化Web应用的目的。

通过介绍，我们知道了Audits非常适合用来分析加载阶段的Web性能，除此之外，Audits还有其他非常实用的功能，比如可以检测我们的代码是否符合一些最佳实践，并给出提示，这样我们就可以根据Audits的提示来决定是否需要优化我们的代码，这个功能非常不错，具体使用方式留给你自己去摸索了。

课后思考

在文中我们又分析Web应用在加载阶段的几个关键时间点，在Audits中，通过对这些时间点的分析，输出了文中介绍的六项性能指标，其实这些时间点也可以通过Performance的时间线(Timelines)来查看，那么今天留给你的任务是：提前熟悉下Performance工具，并对照这文中加载阶段的几个时间点来熟悉下Performance的时间线(Timelines)，欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

你好，我是李兵。

作为一名前端工程师，除了需要编写功能性的代码以外，我们还需要关注Web应用的性能问题，我们应该有能力让我们的Web应用占用最小的资源，并以最高性能运行，这也是前端工程师进阶的必要能力。既然性能这么重要，那么我们今天要来聊聊Web性能问题。

到底什么是Web性能？

我们看下wiki对Web 性能的定义：

> Web 性能描述了Web应用在浏览器上的加载和显示的速度。

因此，当我们讨论Web性能时，其实就是讨论Web应用速度，关于Web应用的速度，我们需要从两个阶段来考虑：

- 页面加载阶段；
- 页面交互阶段。

在本文中，我们会将焦点放到第一个阶段：页面加载阶段的性能，在下篇文章中，我们会来重点分析页面交互阶段的性能。

性能检测工具：Performance vs Audits

要想优化Web的性能，我们就需要有监控Web应用的性能数据，那怎么监控呢？

如果没有工具来模拟各种不同的场景并统计各种性能指标，那么定位Web应用的性能瓶颈将是一件非常困难的任务。幸好，Chrome为我们提供了非常完善的性能检测工具：**Performance**和**Audits**，它们能够准确统计页面在加载阶段和运行阶段的一些核心数据，诸如任务执行记录、首屏展示花费的时长等，有了这些数据我们就能很容易定位到Web应用的**性能瓶颈** 。

首先Performance非常强大，因为它为我们提供了非常多的运行时数据，利用这些数据我们就可以分析出来Web应用的瓶颈。但是要完全学会其使用方式却是非常有难度的，其难点在于这些数据涉及到了特别多的概念，而这些概念又和浏览器的系统架构、消息循环机制、渲染流水线等知识紧密联系在了一起。

相反，Audtis就简单了许多，它将检测到的细节数据隐藏在背后，只提供给我们一些直观的性能数据，同时，还会给我们提供一些优化建议。

Perfomance能让我们看到更多细节数据，但是更加复杂，Audits就比较智能，但是隐藏了更多细节。为了能够让你循序渐进地理解内容，所以本节我们先从简单的Audits入手，看看如何利用它来检测和优化页面在加载阶段的性能，然后在下一节我们再来分析Perfomance。

检测之前准备工作

不过在检测Web的性能指标之前，我们还要配置好工作环境，具体地讲，你需要准备以下内容：

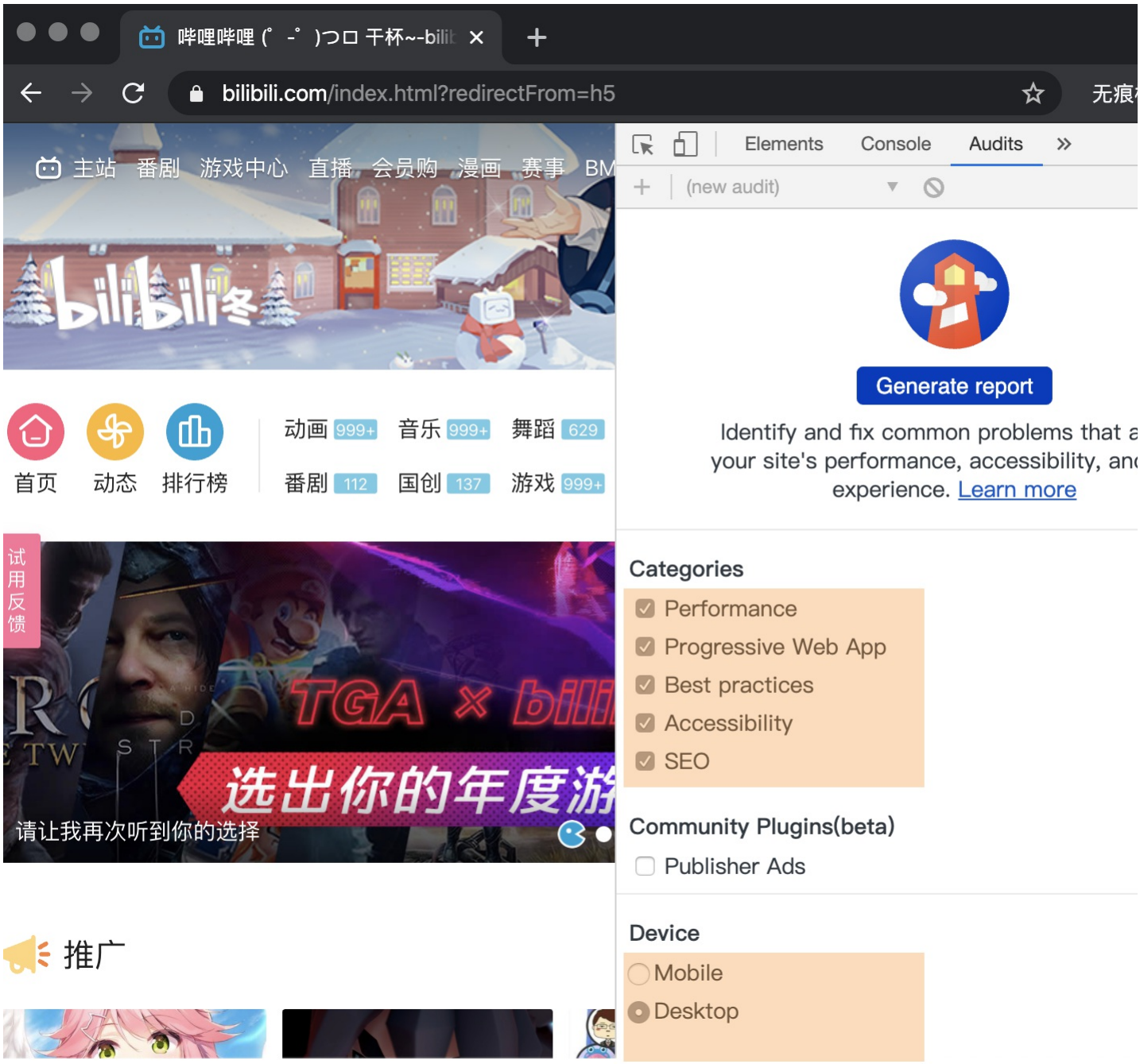
- 首先准备Chrome Canary版的浏览器，Chrome Canary是采用最新技术构建的，它的开发者工具和浏览器特性都是最新的，所以我推荐你使用Chrome Canary来做性能分析。当然你也可以使用稳定版的Chrome。
- 然后我们需要在Chrome的隐身模式下工作，这样可以确保我们安装的扩展、浏览器缓存、Cookie等数据不会影响到检测结果。

利用Audits生成Web性能报告

环境准备好了之后，我们就可以生成站点在加载阶段的性能报告了，这里我们可以拿B站作为分析的列子。

- 首先我们打开浏览器的隐身窗口，Windows系统下面的快捷键是Control+Shift+N，Mac系统下面的快捷键是Command+Shift+N。
- 然后在隐身窗口中输入B站的网站。
- 打开Chrome的开发者工具，选择Audits标签。

最终打开的页面如下图所示：



Audits界面

观察上图中的Audits界面，我们可以看到，在生成报告之前，我们需要先配置Audits，配置模块主要有两部分组成，一个是**监测类型(Categories)**，另外一个是**设备类型(Device)**。

监控类型(Categories)是指需要监控哪些内容，这里有五个对应的选项，它们的功能分别是：

- 监测并分析Web性能(**Performance**)；
- 监测并分析PWA(**Progressive Web App**)程序的性能；
- 监测并分析Web应用是否采用了最佳实践策略(**Best practices**)；
- 监测并分析是否实施了无障碍功能(**Accessibility**)，无障碍功能让一些身体有障碍的人可以方便地浏览你的Web应用。
- 监测并分析Web应用是否采实施了SEO搜索引擎优化(**SEO**)。

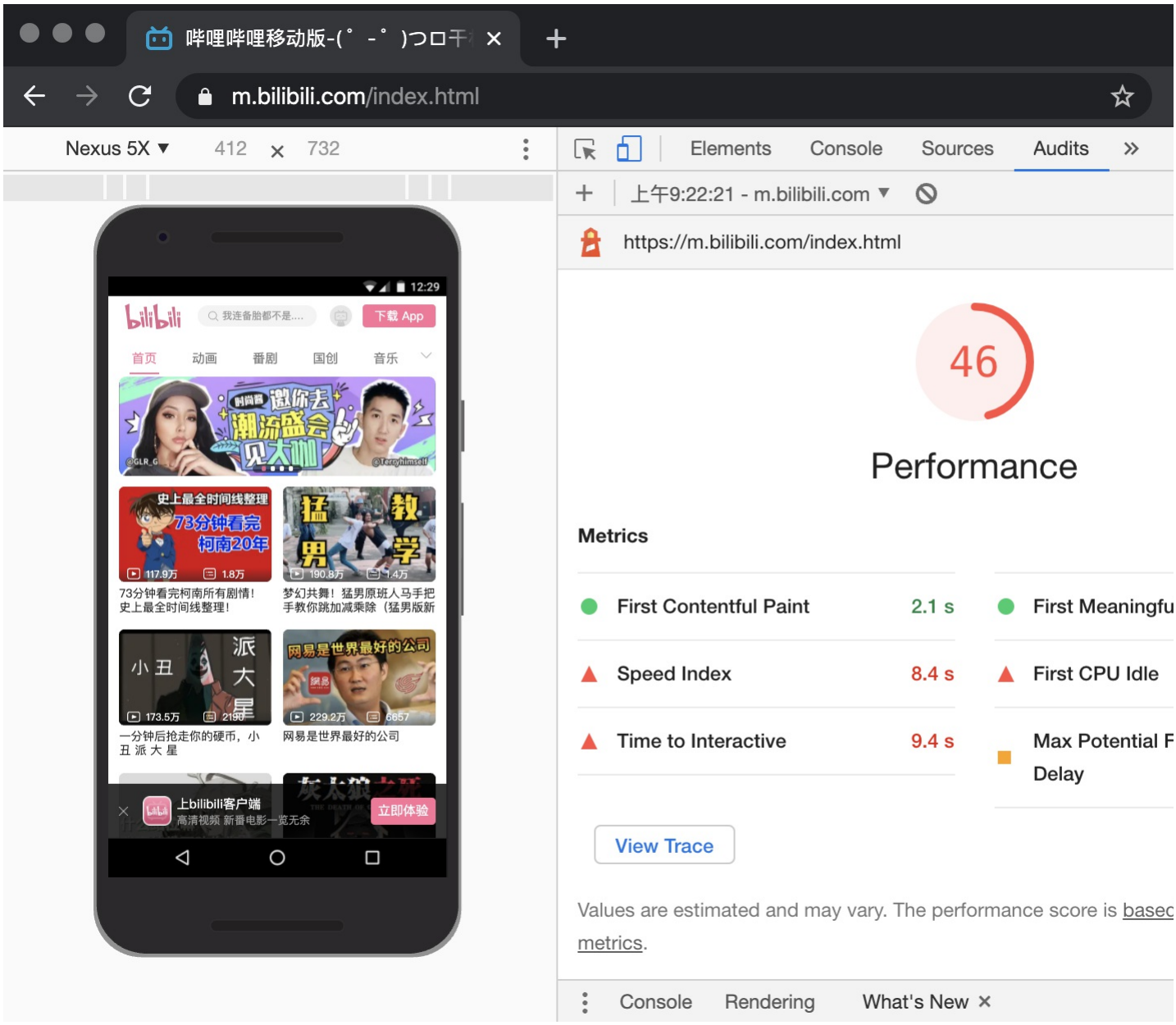
本文我们只需要关注Web应用的加载性能，所以勾选第一个Performance选项就可以了。

再看看**设备(Device)部分**，它给了我们两个选项，Moblie选项是用来模拟移动设备环境的，另外一个Desktop选项是用来模拟桌面环境的。这里我们选择移动设备选项，因为目前大多数流量都是由移动设备产生的，所以移动设备上的Web性能显得更加重要。

配置好选项之后，我们就可以点击最上面的生成报告(Generate report)按钮来生成报告了。

解读性能报告

点击生成报告的按钮之后，我们大约需要等待一分钟左右，Audits就可以生成最终的分析报告了，如下图所示：



生成的报告图

观察上图的分析报告，中间圆圈中的数字表示该站点在加载过程中的总体Web性能得分，总分是100分。我们目前的得分为46分，这表示该站点加载阶段的性能还有很大的提升空间。

Audits除了生成性能指标以外，还会分析该站点并提供了很多优化建议，我们可以根据这些建议来改进Web应用以获得更高的得分，进而获得更好的用户体验效果。

既能分析Web性能得分又能给出优化建议，所以Audits的分析报告还是非常有价值的，那么接下来，我们就来解读下Audits生成的性能报告。

报告的第一个部分是**性能指标(Metrics)**，如下图所示：

Metrics

First Contentful Paint	2.1 s	First Meaningful Paint	2.1 s
Speed Index	8.4 s	First CPU Idle	7.2 s
Time to Interactive	9.4 s	Max Potential First Input Delay	230 ms

View Trace

Values are estimated and may vary. The performance score is based only on these metrics.

性能指标

观察上图，我们可以发现性能指标下面一共有六项内容，这六项内容分别对应了从Web应用的加载到页面展示完成的这段时间中，各个阶段所消耗的时长。在中间还有一个View Trace按钮，点击该按钮可以跳转到Performance标签，并且查看这些阶段在Performance中所对应的位置。最下方是加载过程中各个时间段的屏幕截图。

报告的第二个部分是**可优化项(Opportunities)**，如下图所示：

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Eliminate render-blocking resources	0.76 s
Preconnect to required origins	0.38 s
Warnings: A preconnect <link> was found for "https://api.bilibili.com" but was not used by the browser. Check that you are using the `crossorigin` attribute properly.	
Properly size images	0.16 s

可优化项(Opportunities)

这些可优化项是Audits发现页面中的一些可以直接优化的部分，你可以对照Audits给的这些提示来优化你的Web应用。

报告的第三部分是**手动诊断(Diagnostics)**，如下图所示：

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- Avoid enormous network payloads — Total size was 4,257 KB
- Serve static assets with an efficient cache policy — 7 resources found
- Avoid an excessive DOM size — 892 elements
- Avoid chaining critical requests — 12 chains found
- Keep request counts low and transfer sizes small — 236 requests • 4,257 KB

Passed audits (16)

手动诊断(Diagnostics)

在手动诊断部分，采集了一些可能存在性能问题的指标，这些指标可能会影响到页面的加载性能，Audits把详情列出来，并让你依据实际情况，来手动排查每一项。

报告的最后一部分是**运行时设置(Runtime Settings)**，如下图所示：

Runtime Settings

URL	https://m.bilibili.com/index.html
Fetch time	Nov 30, 2019, 9:22 AM GMT+8
Device	Emulated Nexus 5X
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3980.0 Safari/537.36
User agent (network)	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Mobile Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1612

Generated by **Lighthouse** 5.7.0 | File an issue

运行时设置(Runtime Settings)

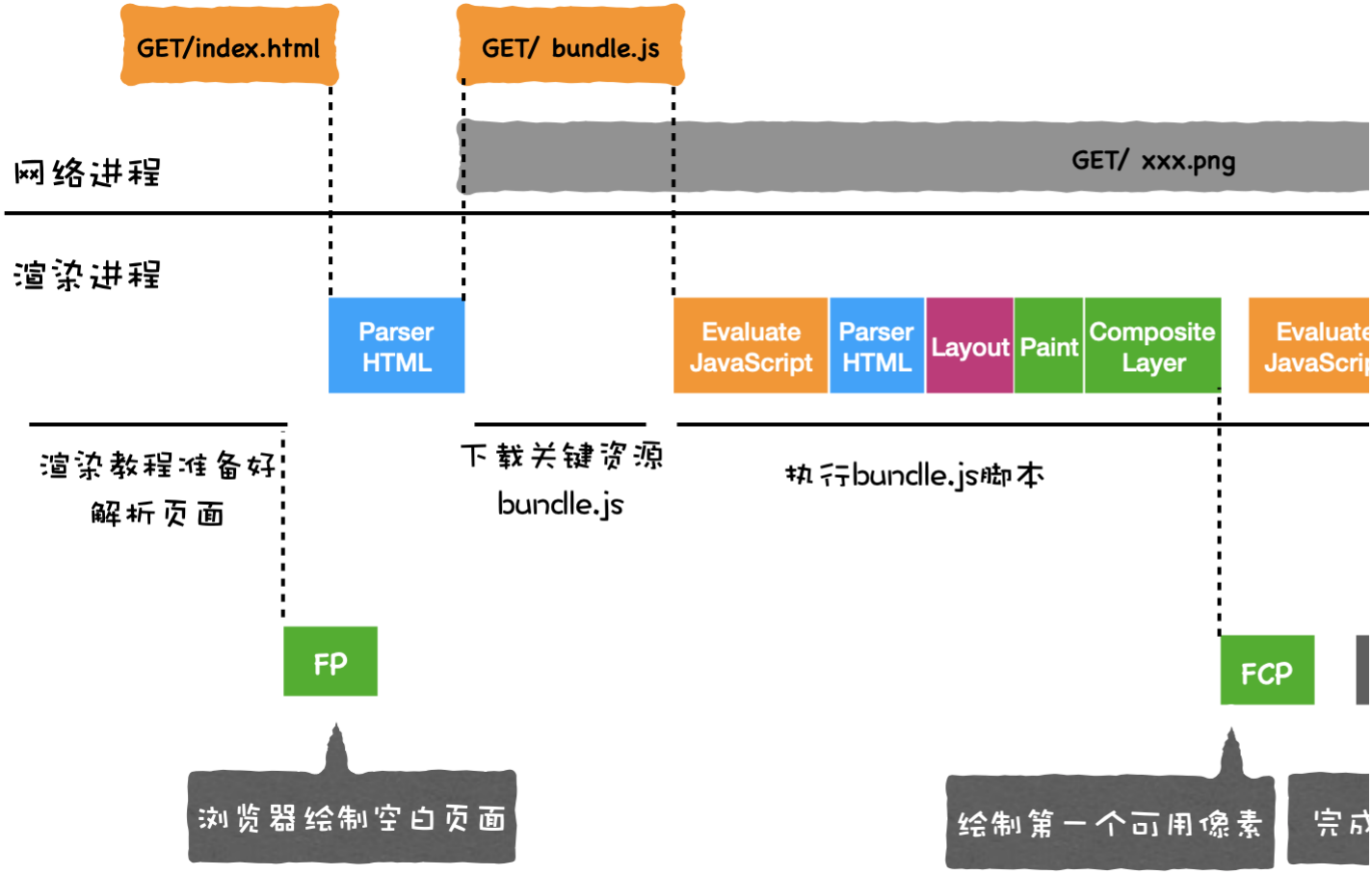
观察上图，这是运行时的一些基本数据，如果选择移动设备模式，你可以看到发送网络请求时的User Agent 会变成设备相关信息，还有会模拟设备的网速，这个体现在网络限速上。

根据性能报告优化Web性能

现在有了性能报告，接下来我们就可以依据报告来分析如何优化Web应用了。最直接的方式是想办法提高性能指标的分数，而性能指标的分数是由六项指标决定的，它们分别是：

1. 首次绘制(First Paint)；
2. 首次有效绘制(First Meaningfull Paint)；
3. 首屏时间(Speed Index)；
4. 首次CPU空闲时间(First CPU Idle)；
5. 完全可交互时间(Time to Interactive)；
6. 最大估计输入延时(Max Potential First Input Delay)。

那么接下来我会逐一分析六项指标的含义，并讨论如何提升这六项指标的数值。这六项都是页面在加载过程中的性能指标，所以要弄明白这六项指标的具体含义，我们还得结合页面的加载过程来分析。一图胜过千言，我们还是先看下面这张页面从加载到展示的过程图：



页面加载过程

观察上图的页面加载过程，我们发现，在渲染进程确认要渲染当前的请求后，渲染进程会创建一个空白页面，我们把创建空白页面的这个时间点称为**First Paint**，简称**FP**。

然后渲染进程继续请求关键资源，我们在《25丨页面性能：如何系统地优化页面？》这节中介绍过了关键资源，并且知道了关键资源包括了JavaScript文件和CSS文件，因为关键资源会阻塞页面的渲染，所以我们需要等待关键资源加载完成后，才能执行进一步的页面绘制。

上图中，bundle.js是关键资源，因此需要完成加载之后，渲染进程才能执行该脚本，然后脚本会修改DOM，引发重绘和重排等一系列操作，当页面中绘制了第一个像素时，我们把这个时间点称为**First Content Paint**，简称**FCP**。

接下来继续执行JavaScript脚本，当首屏内容完全绘制完成时，我们把这个时间点称为**Largest Content Paint**，简称**LCP**。

在FCP和LCP中间，还有一个FMP，这个是首次有效绘制，由于FMP计算复杂，而且容易出错，现在不推荐使用该指标，所以这里我们也不做过多介绍了。

接下来JavaScript脚本执行结束，渲染进程判断该页面的DOM生成完毕，于是触发DOMContentLoad事件。等所有资源都加载结束之后，再触发onload事件。

好了，以上就是页面在加载过程中各个重要的时间节点，了解了这些时间节点，我们就可以来聊聊性能报告的六项指标的含义并讨论如何优化这些指标。

我们先来分析下**第一项指标FP**，如果FP时间过久，那么直接说明了一个问题，那就是页面的HTML文件可能由于网络原因导致加载时间过久，这块具体的分析过程你可以参考《21丨Chrome开发者工具：利用网络面板做性能分析》这节内容。

第二项是FMP，上面也提到过由于FMP计算复杂，所以现在不建议使用该指标了，另外由于LCP的计算规则简单，所以推荐使用LCP指标，具体文章你可以参考这里。不过是FMP还是LCP，优化它们的方式都是类似的，你可以结合上图，如果FMP和LCP消耗时间过久，那么有可能是加载关键资源花的时间过久，也有可能是JavaScript执行过程中所花的时间过久，所以我们可以针对具体的情况来具体分析。

第三项是首屏时间(Speed Index)，这就是我们上面提到的LCP，它表示填满首屏页面所消耗的时间，首屏时间的值越大，那么加载速度越慢，具体的优化方式同优化第二项FMP是一样。

第四项是首次CPU空闲时间(First CPU Idle)，也称为First Interactive，它表示页面达到最小化可交互的时间，也就是说并不需要等到页面上的所有元素都可交互，只要可以对大部分用户输入做出响应即可。要缩短首次CPU空闲时长，我们就需要尽可能快地加载完关键资源，尽可能快地渲染出来首屏内容，因此优化方式和第二项FMP和第三项LCP是一样的。

第五项是完全可交互时间(Time to Interactive)，简称TTI，它表示页面中所有元素都达到了可交互的时长。简单理解就这时候页面的内容已经完全显示出来了，所有的JavaScript事件已经注册完成，页面能够对用户的交互做出快速响应，通常满足响应速度在50毫秒以内。如果要解决TTI时间过久的问题，我们可以推迟执行一些和生成页面无关的JavaScript工作。

第六项是最大估计输入延时(Max Potential First Input Delay），这个指标是估计你的Web页面在加载最繁忙的阶段， 窗口中响应用户输入所需的时间，为了改善该指标，我们可以使用WebWorker来执行一些计算，从而释放主线程。另一个有用的措施是重构CSS选择器，以确保它们执行较少的计算。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容：

本文我们主要讨论如何优化加载阶段的Web应用的性能。

要想优化Web性能，首先得需要有Web应用的性能数据。所以接下来，我们介绍了Chrome采集Web性能数据的两个工具：Performance和Audits，Performance可以采集非常多的性能，但是其使用难度大，相反，Audtis就简单了许多，它会分析检测到的性能数据并给出站点的性能得分，同时，还会给我们提供一些优化建议。

我们先从简单的工具上手，所以本文我们主要分析了Audits的使用方式，先介绍了如何使用Audits生成性能报告，然后我们解读了性能报告中的每一项内容。

大致了解Audits生成的性能报告之后，我们又分析Web应用在加载阶段的几个关键时间点，最后我们分析性能指标的具体含义以及如何提高性能指标的分数，从而达到优化Web应用的目的。

通过介绍，我们知道了Audits非常适合用来分析加载阶段的Web性能，除此之外，Audits还有其他非常实用的功能，比如可以检测我们的代码是否符合一些最佳实践，并给出提示，这样我们就可以根据Audits的提示来决定是否需要优化我们的代码，这个功能非常不错，具体使用方式留给你自己去摸索了。

课后思考

在文中我们又分析Web应用在加载阶段的几个关键时间点，在Audits中，通过对这些时间点的分析，输出了文中介绍的六项性能指标，其实这些时间点也可以通过Performance的时间线(Timelines)来查看，那么今天留给你的任务是：提前熟悉下Performance工具，并对照这文中加载阶段的几个时间点来熟悉下Performance的时间线(Timelines)，欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。

你好，我是李兵。

作为一名前端工程师，除了需要编写功能性的代码以外，我们还需要关注Web应用的性能问题，我们应该有能力让我们的Web应用占用最小的资源，并以最高性能运行，这也是前端工程师进阶的必要能力。既然性能这么重要，那么我们今天要来聊聊Web性能问题。

到底什么是Web性能？

我们看下wiki对Web 性能的定义：

> Web 性能描述了Web应用在浏览器上的加载和显示的速度。

因此，当我们讨论Web性能时，其实就是讨论Web应用速度，关于Web应用的速度，我们需要从两个阶段来考虑：

- 页面加载阶段；
- 页面交互阶段。

在本文中，我们会将焦点放到第一个阶段：页面加载阶段的性能，在下篇文章中，我们会来重点分析页面交互阶段的性能。

性能检测工具：Performance vs Audits

要想优化Web的性能，我们就需要有监控Web应用的性能数据，那怎么监控呢？

如果没有工具来模拟各种不同的场景并统计各种性能指标，那么定位Web应用的性能瓶颈将是一件非常困难的任务。幸好，Chrome为我们提供了非常完善的性能检测工具：**Performance**和**Audits**，它们能够准确统计页面在加载阶段和运行阶段的一些核心数据，诸如任务执行记录、首屏展示花费的时长等，有了这些数据我们就能很容易定位到Web应用的**性能瓶颈** 。

首先Performance非常强大，因为它为我们提供了非常多的运行时数据，利用这些数据我们就可以分析出来Web应用的瓶颈。但是要完全学会其使用方式却是非常有难度的，其难点在于这些数据涉及到了特别多的概念，而这些概念又和浏览器的系统架构、消息循环机制、渲染流水线等知识紧密联系在了一起。

相反，Audtis就简单了许多，它将检测到的细节数据隐藏在背后，只提供给我们一些直观的性能数据，同时，还会给我们提供一些优化建议。

Perfomance能让我们看到更多细节数据，但是更加复杂，Audits就比较智能，但是隐藏了更多细节。为了能够让你循序渐进地理解内容，所以本节我们先从简单的Audits入手，看看如何利用它来检测和优化页面在加载阶段的性能，然后在下一节我们再来分析Perfomance。

检测之前准备工作

不过在检测Web的性能指标之前，我们还要配置好工作环境，具体地讲，你需要准备以下内容：

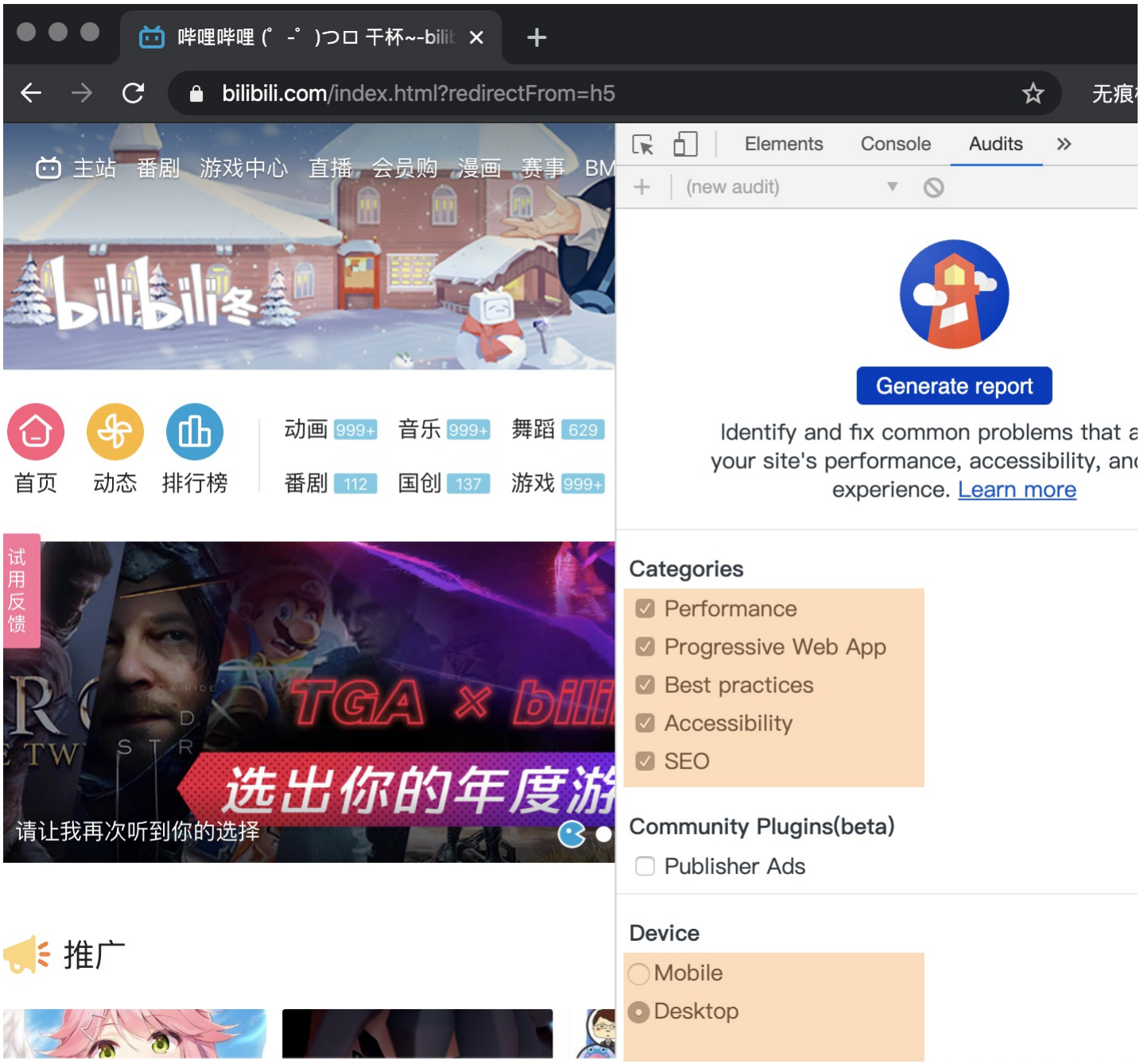
- 首先准备Chrome Canary版的浏览器，Chrome Canary是采用最新技术构建的，它的开发者工具和浏览器特性都是最新的，所以我推荐你使用Chrome Canary来做性能分析。当然你也可以使用稳定版的Chrome。
- 然后我们需要在Chrome的隐身模式下工作，这样可以确保我们安装的扩展、浏览器缓存、Cookie等数据不会影响到检测结果。

利用Audits生成Web性能报告

环境准备好了之后，我们就可以生成站点在加载阶段的性能报告了，这里我们可以拿B站作为分析的列子。

- 首先我们打开浏览器的隐身窗口，Windows系统下面的快捷键是Control+Shift+N，Mac系统下面的快捷键是Command+Shift+N。
- 然后在隐身窗口中输入B站的网站。
- 打开Chrome的开发者工具，选择Audits标签。

最终打开的页面如下图所示：



Audits界面

观察上图中的Audits界面，我们可以看到，在生成报告之前，我们需要先配置Audits，配置模块主要有两部分组成，一个是**监测类型(Categories)**，另外一个是**设备类型(Device)**。

监控类型(Categories)是指需要监控哪些内容，这里有五个对应的选项，它们的功能分别是：

- 监测并分析Web性能(**Performance**)；
- 监测并分析PWA(**Progressive Web App**)程序的性能；
- 监测并分析Web应用是否采用了最佳实践策略(**Best practices**)；
- 监测并分析是否实施了无障碍功能(**Accessibility**)，无障碍功能让一些身体有障碍的人可以方便地浏览你的Web应用。
- 监测并分析Web应用是否采实施了SEO搜索引擎优化(**SEO**)。

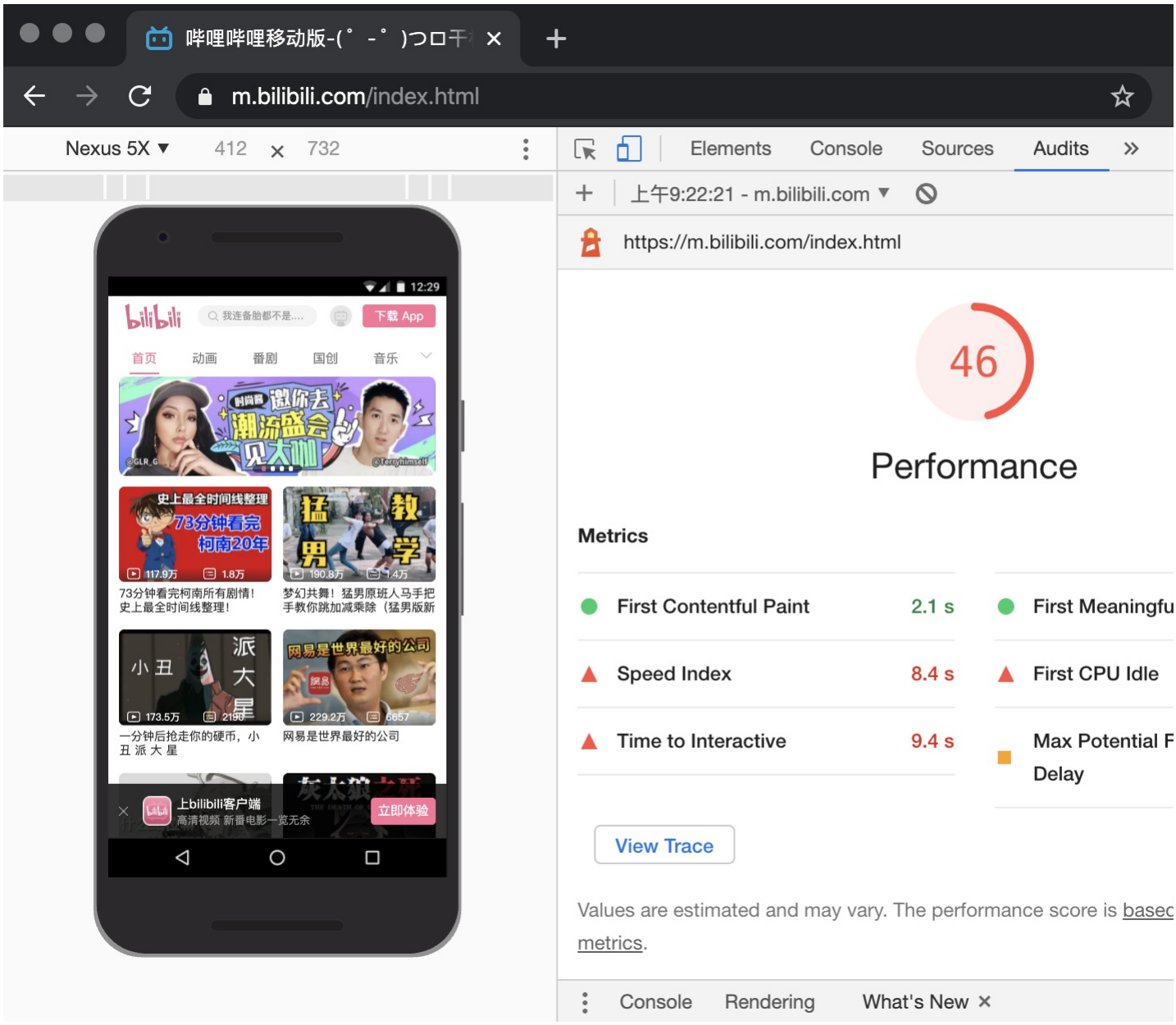
本文我们只需要关注Web应用的加载性能，所以勾选第一个Performance选项就可以了。

再看看**设备(Device)部分**，它给了我们两个选项，Moblie选项是用来模拟移动设备环境的，另外一个Desktop选项是用来模拟桌面环境的。这里我们选择移动设备选项，因为目前大多数流量都是由移动设备产生的，所以移动设备上的Web性能显得更加重要。

配置好选项之后，我们就可以点击最上面的生成报告(Generate report)按钮来生成报告了。

解读性能报告

点击生成报告的按钮之后，我们大约需要等待一分钟左右，Audits就可以生成最终的分析报告了，如下图所示：



生成的报告图

观察上图的分析报告，中间圆圈中的数字表示该站点在加载过程中的总体Web性能得分，总分是100分。我们目前的得分为46分，这表示该站点加载阶段的性能还有很大的提升空间。

Audits除了生成性能指标以外，还会分析该站点并提供了很多优化建议，我们可以根据这些建议来改进Web应用以获得更高的得分，进而获得更好的用户体验效果。

既能分析Web性能得分又能给出优化建议，所以Audits的分析报告还是非常有价值的，那么接下来，我们就来解读下Audits生成的性能报告。

报告的第一个部分是**性能指标(Metrics)**，如下图所示：

Metrics

First Contentful Paint	2.1 s	First Meaningful Paint	2.1 s
Speed Index	8.4 s	First CPU Idle	7.2 s
Time to Interactive	9.4 s	Max Potential First Input Delay	230 ms

View Trace

Values are estimated and may vary. The performance score is based only on these metrics.

性能指标

观察上图，我们可以发现性能指标下面一共有六项内容，这六项内容分别对应了从Web应用的加载到页面展示完成的这段时间中，各个阶段所消耗的时长。在中间还有一个View Trace按钮，点击该按钮可以跳转到Performance标签，并且查看这些阶段在Performance中所对应的位置。最下方是加载过程中各个时间段的屏幕截图。

报告的第二个部分是**可优化项(Opportunities)**，如下图所示：

Opportunities — These suggestions can help your page load faster. They don't directly affect the Performance score.

Opportunity	Estimated Savings
Eliminate render-blocking resources	0.76 s
Preconnect to required origins	0.38 s
Warnings: A preconnect <link> was found for "https://api.bilibili.com" but was not used by the browser. Check that you are using the `crossorigin` attribute properly.	
Properly size images	0.16 s

可优化项(Opportunities)

这些可优化项是Audits发现页面中的一些可以直接优化的部分，你可以对照Audits给的这些提示来优化你的Web应用。

报告的第三部分是**手动诊断(Diagnostics)**，如下图所示：

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- Avoid enormous network payloads — Total size was 4,257 KB
- Serve static assets with an efficient cache policy — 7 resources found
- Avoid an excessive DOM size — 892 elements
- Avoid chaining critical requests — 12 chains found
- Keep request counts low and transfer sizes small — 236 requests • 4,257 KB

Passed audits (16)

手动诊断(Diagnostics)

在手动诊断部分，采集了一些可能存在性能问题的指标，这些指标可能会影响到页面的加载性能，Audits把详情列出来，并让你依据实际情况，来手动排查每一项。

报告的最后一部分是**运行时设置(Runtime Settings)**，如下图所示：

Runtime Settings

URL	https://m.bilibili.com/index.html
Fetch time	Nov 30, 2019, 9:22 AM GMT+8
Device	Emulated Nexus 5X
Network throttling	150 ms TCP RTT, 1,638.4 Kbps throughput (Simulated)
CPU throttling	4x slowdown (Simulated)
User agent (host)	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3980.0 Safari/537.36
User agent (network)	Mozilla/5.0 (Linux; Android 6.0.1; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3694.0 Mobile Safari/537.36 Chrome-Lighthouse
CPU/Memory Power	1612

Generated by **Lighthouse** 5.7.0 | File an issue

运行时设置(Runtime Settings)

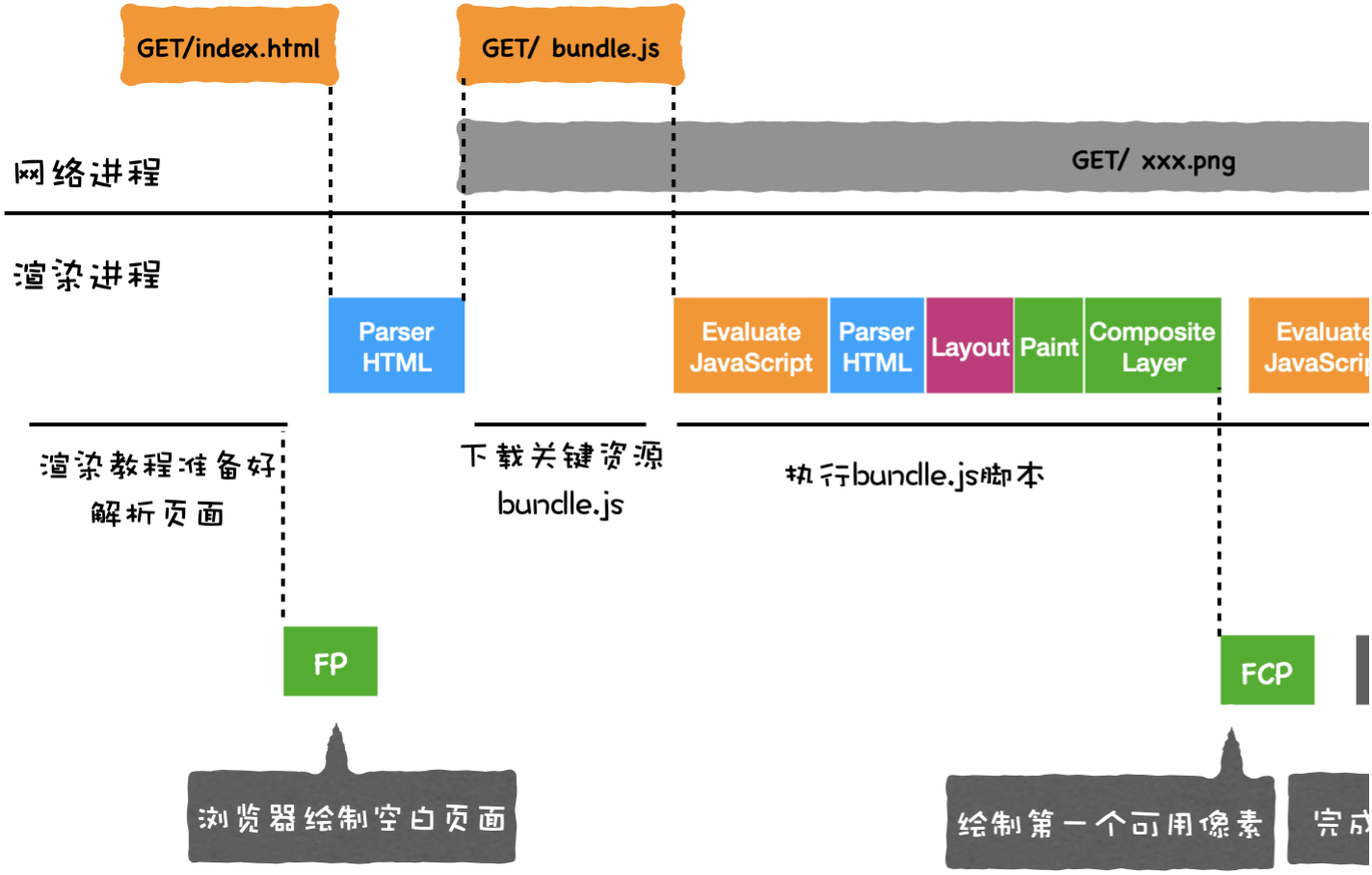
观察上图，这是运行时的一些基本数据，如果选择移动设备模式，你可以看到发送网络请求时的User Agent 会变成设备相关信息，还有会模拟设备的网速，这个体现在网络限速上。

根据性能报告优化Web性能

现在有了性能报告，接下来我们就可以依据报告来分析如何优化Web应用了。最直接的方式是想办法提高性能指标的分数，而性能指标的分数是由六项指标决定的，它们分别是：

1. 首次绘制(First Paint)；
2. 首次有效绘制(First Meaningfull Paint)；
3. 首屏时间(Speed Index)；
4. 首次CPU空闲时间(First CPU Idle)；
5. 完全可交互时间(Time to Interactive)；
6. 最大估计输入延时(Max Potential First Input Delay)。

那么接下来我会逐一分析六项指标的含义，并讨论如何提升这六项指标的数值。这六项都是页面在加载过程中的性能指标，所以要弄明白这六项指标的具体含义，我们还得结合页面的加载过程来分析。一图胜过千言，我们还是先看下面这张页面从加载到展示的过程图：



页面加载过程

观察上图的页面加载过程，我们发现，在渲染进程确认要渲染当前的请求后，渲染进程会创建一个空白页面，我们把创建空白页面的这个时间点称为**First Paint**，简称**FP**。

然后渲染进程继续请求关键资源，我们在《25丨页面性能：如何系统地优化页面？》这节中介绍过了关键资源，并且知道了关键资源包括了JavaScript文件和CSS文件，因为关键资源会阻塞页面的渲染，所以我们需要等待关键资源加载完成后，才能执行进一步的页面绘制。

上图中，bundle.js是关键资源，因此需要完成加载之后，渲染进程才能执行该脚本，然后脚本会修改DOM，引发重绘和重排等一系列操作，当页面中绘制了第一个像素时，我们把这个时间点称为**First Content Paint**，简称**FCP**。

接下来继续执行JavaScript脚本，当首屏内容完全绘制完成时，我们把这个时间点称为**Largest Content Paint**，简称**LCP**。

在FCP和LCP中间，还有一个FMP，这个是首次有效绘制，由于FMP计算复杂，而且容易出错，现在不推荐使用该指标，所以这里我们也不做过多介绍了。

接下来JavaScript脚本执行结束，渲染进程判断该页面的DOM生成完毕，于是触发DOMContentLoad事件。等所有资源都加载结束之后，再触发onload事件。

好了，以上就是页面在加载过程中各个重要的时间节点，了解了这些时间节点，我们就可以来聊聊性能报告的六项指标的含义并讨论如何优化这些指标。

我们先来分析下**第一项指标FP**，如果FP时间过久，那么直接说明了一个问题，那就是页面的HTML文件可能由于网络原因导致加载时间过久，这块具体的分析过程你可以参考《21丨Chrome开发者工具：利用网络面板做性能分析》这节内容。

第二项是FMP，上面也提到过由于FMP计算复杂，所以现在不建议使用该指标了，另外由于LCP的计算规则简单，所以推荐使用LCP指标，具体文章你可以参考这里。不过是FMP还是LCP，优化它们的方式都是类似的，你可以结合上图，如果FMP和LCP消耗时间过久，那么有可能是加载关键资源花的时间过久，也有可能是JavaScript执行过程中所花的时间过久，所以我们可以针对具体的情况来具体分析。

第三项是首屏时间(Speed Index)，这就是我们上面提到的LCP，它表示填满首屏页面所消耗的时间，首屏时间的值越大，那么加载速度越慢，具体的优化方式同优化第二项FMP是一样。

第四项是首次CPU空闲时间(First CPU Idle)，也称为First Interactive，它表示页面达到最小化可交互的时间，也就是说并不需要等到页面上的所有元素都可交互，只要可以对大部分用户输入做出响应即可。要缩短首次CPU空闲时长，我们就需要尽可能快地加载完关键资源，尽可能快地渲染出来首屏内容，因此优化方式和第二项FMP和第三项LCP是一样的。

第五项是完全可交互时间(Time to Interactive)，简称TTI，它表示页面中所有元素都达到了可交互的时长。简单理解就这时候页面的内容已经完全显示出来了，所有的JavaScript事件已经注册完成，页面能够对用户的交互做出快速响应，通常满足响应速度在50毫秒以内。如果要解决TTI时间过久的问题，我们可以推迟执行一些和生成页面无关的JavaScript工作。

第六项是最大估计输入延时(Max Potential First Input Delay），这个指标是估计你的Web页面在加载最繁忙的阶段， 窗口中响应用户输入所需的时间，为了改善该指标，我们可以使用WebWorker来执行一些计算，从而释放主线程。另一个有用的措施是重构CSS选择器，以确保它们执行较少的计算。

总结

好了，今天的内容就介绍到这里，下面我来总结下本文的主要内容：

本文我们主要讨论如何优化加载阶段的Web应用的性能。

要想优化Web性能，首先得需要有Web应用的性能数据。所以接下来，我们介绍了Chrome采集Web性能数据的两个工具：Performance和Audits，Performance可以采集非常多的性能，但是其使用难度大，相反，Audtis就简单了许多，它会分析检测到的性能数据并给出站点的性能得分，同时，还会给我们提供一些优化建议。

我们先从简单的工具上手，所以本文我们主要分析了Audits的使用方式，先介绍了如何使用Audits生成性能报告，然后我们解读了性能报告中的每一项内容。

大致了解Audits生成的性能报告之后，我们又分析Web应用在加载阶段的几个关键时间点，最后我们分析性能指标的具体含义以及如何提高性能指标的分数，从而达到优化Web应用的目的。

通过介绍，我们知道了Audits非常适合用来分析加载阶段的Web性能，除此之外，Audits还有其他非常实用的功能，比如可以检测我们的代码是否符合一些最佳实践，并给出提示，这样我们就可以根据Audits的提示来决定是否需要优化我们的代码，这个功能非常不错，具体使用方式留给你自己去摸索了。

课后思考

在文中我们又分析Web应用在加载阶段的几个关键时间点，在Audits中，通过对这些时间点的分析，输出了文中介绍的六项性能指标，其实这些时间点也可以通过Performance的时间线(Timelines)来查看，那么今天留给你的任务是：提前熟悉下Performance工具，并对照这文中加载阶段的几个时间点来熟悉下Performance的时间线(Timelines)，欢迎在留言区分享你的想法。

感谢阅读，如果你觉得这篇文章对你有帮助的话，也欢迎把它分享给更多的朋友。