

1.1 引例描述



初代产品完成后应对 Web 页面进行响应测试，通过改变视窗大小（例如缩放浏览器）查看页面的组成、跳转、兼容性等。Google Chrome 是由 Google 公司开发的一款网页浏览器，自 2008 年 9 月第一个测试版本发布以来，其市场占有率逐步上升至 2016 年 5 月，Chrome 已超越 Firefox 成为全球市场占有率第二的浏览器。Chrome 的受欢迎程度与其优秀的性能与兼容性密不可分，并且越来越多的网络应用程序都添加了对 Chrome 的支持，也足以体现网络应用的开发人员对 Chrome 的认可与青睐，而其中最重要的原因之一，莫过于 Chrome 所提供的强大的开发者工具。诸多强大的功能模块，适应不同场合的需要。

1.2 任务陈述

表 4-1 学习任务单

一、达成目标
1. 掌握查看 Element、Network，能够调试 JavaScript，用 console 显示需要的信息；
2. 能够结合 Chrome 工具验证三个小任务中的解决方案；
二、学习任务
1.应用 Emulation 模式调试设备 ,用 Network、Sensor 等选项模拟查看已实现网页的效果；
2.仔细练习并掌握在 Source 中调试 JavaScript 的方法；
3.以小组的方式互相检测对方完成的页面，记录发现的问题并展开讨论；
4.针对 Cookie 相关知识，尝试搭建一个服务器来实现前后台交互；
三、参考资源
1. 在 https://developers.google.cn/web/tools/chrome-devtools/ 寻找更多信息；
2. 搜索 Chrome 功能强大的插件,如尝试更换皮肤、增加助手、加上二维码等等；
四、困惑与建议
本单元中涉及 JavaScript 调试部分的内容可以结合后续 DOM 单元一起学习，请将遇到的难点和不清晰的地方通过截屏等方式记录，准备与教师面对面讨论问题。

1.3 知识准备

1947 年 9 月 9 日下午 3 点 45 分，Grace Murray Hopper 在她的记录本上记下了一个计算机 Bug——在 Harvard Mark II 计算机里找到的一只飞蛾 她把飞蛾贴在日记本上，



并写道“ First actual case of bug being found”。

本单元将通过 Chrome 浏览器提供的开发者工具来演示如何调试一个项目，可以通过更多工具选项->开发者工具即可进入调试界面，常通过点击键盘上的“F12” 按键直接进入调试界面。

在不涉及与服务器交互情况下，将演示如何对项目中的“缺陷”的地方进行微调（该调整最终不会修改源文件，效果也仅限于当前浏览器，刷新就重置了）。并讲解如何通过 Elements 工具微调网页中的 CSS 属性、编辑网页代码、调整 DOM 树结构、查看目标网页的代码设计命名规范，学习目标网页中良好的设计结构和漂亮的 CSS 特效设计。如何通过 Console 工具去调试浏览器提供的方法配合 log 信息查看 JavaScript 的运行流程，通过 Source 工具去调试 web 应用的逻辑，查看事件与每个变量值，如何通过 Network 工具抓取网络请求连接资源和详细信息等。

1.3.1 知识点 1 Chrome 调试工具

第一次进入 Chrome 调试工具后，默认看到的是 Elements 选项，它分为 2 大区域，这里把它命名为 A 区和 B 区，如下图所示：



在 Element 选项中主要分为两大部分，也就是上图中所标注的 A 区和 B 区，以下：

A 区是页面 DOM 树结构化图形界面，B 区是针对 A 区中的元素/标签的详细说明（比如 DOM 样式，结构，事件等）。在 A 区中，每当你的鼠标移动到任意一个元素/标签上，对应的 HTML 视图中会给元素加上蓝色背景，如下图所示：



调试工具界面 2

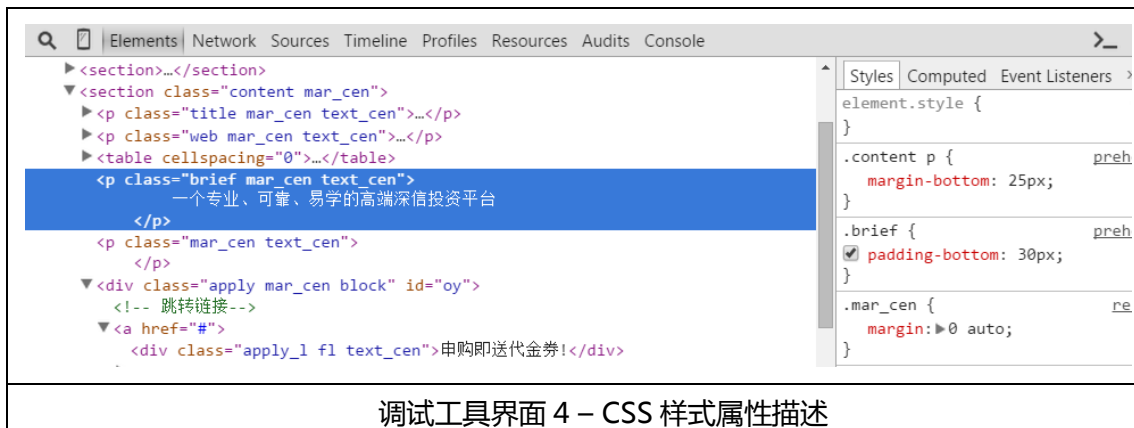
因为背景图片也是蓝色的原因导致，虽然 HTML 视图上的蓝色不明显，但还可以看到它有其它 2 种颜色，分别是绿色和橙色，其中绿色代表的是该元素的内间距，橙色代表的是该元素的外边距，通过 B 区中的这个面板也可以很清晰的明白其中的具体数值，如下图所示：



调试工具界面 3 – 元素盒子模型

注意到 Element 选项中最下面的一栏，当单击选中的元素，会显示该元素/标签在 HTML 结构中的位置关系，比如上图中就明确告知选中的 p 标签处理在某个 section 标签下。

同时还可以在 B 区中点击 Style 选项编辑该元素的 CSS 样式，并且看到 HTML 视图的实时更新，如下图所示：



调试工具界面 4 – CSS 样式属性描述

通过 A 区中点击的元素，可以在 B 区中查看它的具体样式描述，上图中的样式描述了该元素/标签的内边距和外边距，它们的属性和之前的图中的盒子模型中所显示的数值一致。

除了查看该元素/标签的样式之外，还可以查看它的 Event Listeners。事件可以在文档 (Document) 结构的任何部分被触发，触发者可以是用户操作，也可以是浏览器本身。事件并不是只是在一处被触发和终止而是在整个 document 中流动，拥有自己的生命周期。而这个生命周期让 DOM 事件有更多的用途和可扩展性。作为一个开发人员，我们必须理解 DOM 事件是如何工作的。

事件(Event)对象在 event 第一次触发的时候被创建出来，并且一直伴随着事件在 DOM 结构中流转的整个生命周期。event 对象会被作为第一个参数传递给事件监听的回调函数。我们可以通过这个 event 对象来获取到大量当前事件相关的信息(更多信息请参阅 W3C)：

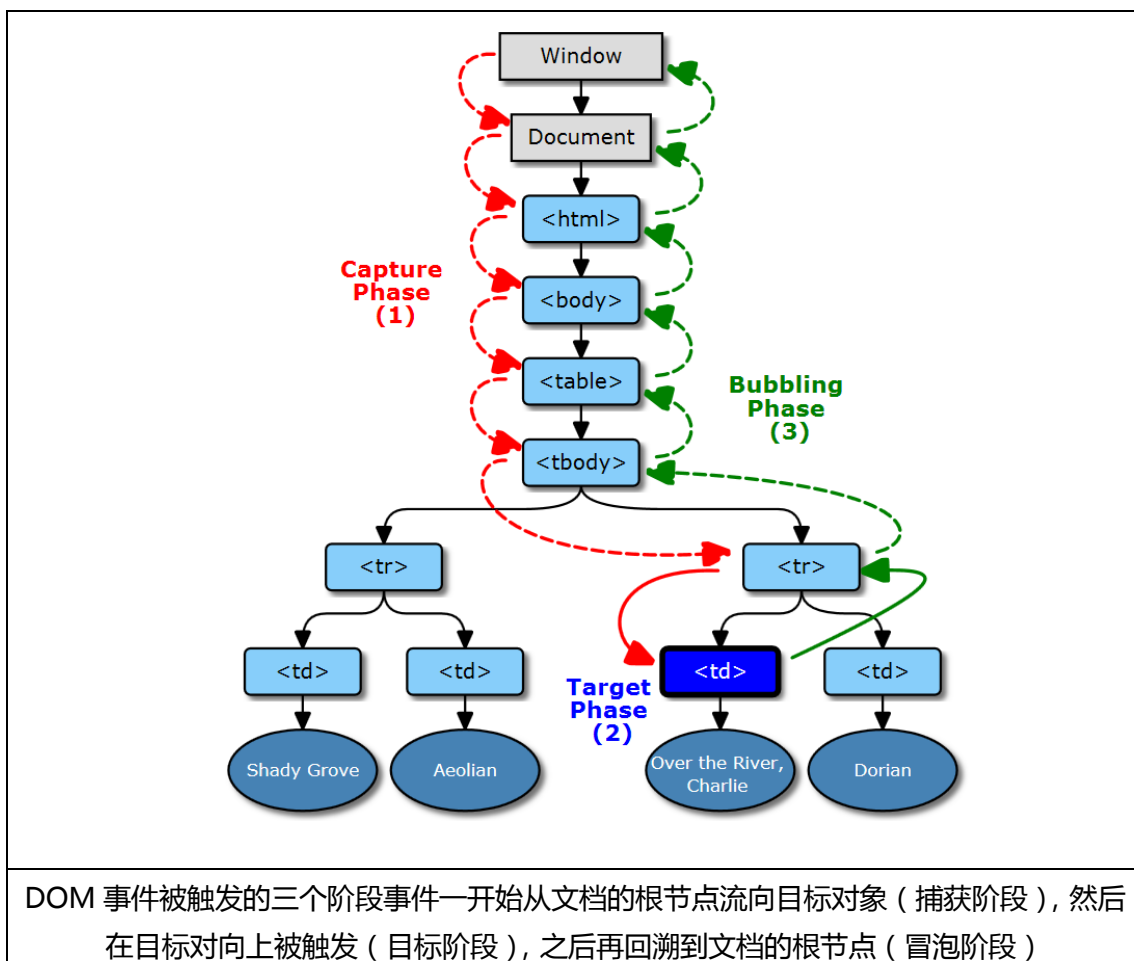
- type (String) — 事件的名称
- target (node) — 事件起源的 DOM 节点
- currentTarget?(node) — 当前回调函数被触发的 DOM 节点
- bubbles (boolean) — 指明这个事件是否是一个冒泡事件。

这需要在 B 区中进行切换，该选项在第 3 个，如下图所示：



调试工具界面 5 – 事件监听查看

上图中选中的元素可以观察到它的事件监听，其中 click 是事件名称，div#oy 是事件索引名称，attachment 是事件来源，handler 是事件回调主体内容，node 是节点内容，useCapture 是表示该事件是否向上冒泡。

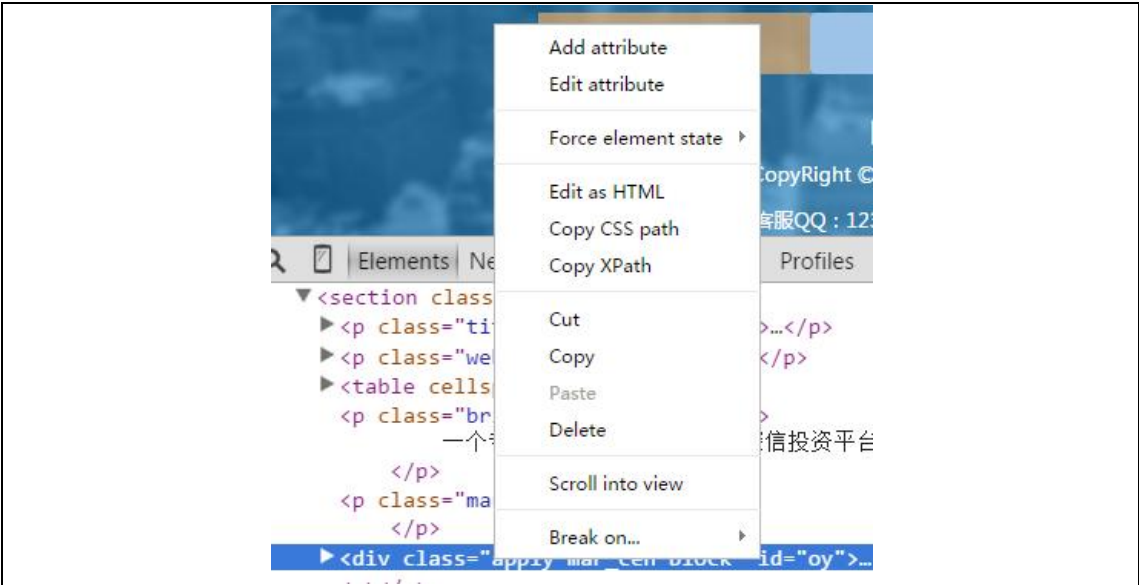


事件的第一个阶段是捕获阶段。事件从文档的根节点出发，随着 DOM 树的结构向事件的目标节点流去。途中经过各个层次的 DOM 节点，并在各节点上触发捕获事件，直到到达事件的目标节点。捕获阶段的主要任务是建立传播路径，在冒泡阶段，事件会通过这个路径回溯到文档根节点。泡过程非常有用。它将从我们对特定元素的事件监听中释放出来，相反，我们可以监听 DOM 树上更上层的元素，等待事件冒泡的到达。比如想全选某个 Div 的内容，则要进行事件冒泡捕获整个父 div 然后选中。

选中任意一个元素，点击鼠标右键，会看到一个窗口出现，里面有若干个选项（因版本变化，出现的选项也会略有变化）。

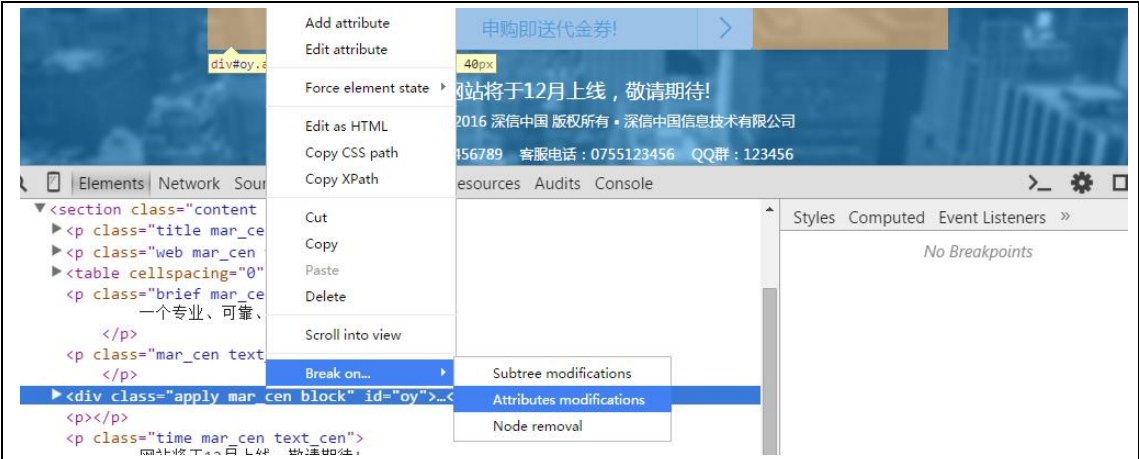
- Add attribute：为该元素添加样式属性；
- Edit attribute：修改该元素的样式属性；
- Force element state：为元素激活某种状态；
- Edit as HTML：编辑该元素，可对该标签代码编辑，复制路径，剪贴，复制等；
- Break on：为该元素添加 DOM 操作事件监听，包含 3 个选项，树结构改变、属性改变、节点移除等；

如下图所示：



调试工具界面 6

这些选项的作用是帮助我们监控和定位操作元素的代码，如下图所示：



调试工具界面 7

点击完毕后，B 区将会出现一个默认打钩的选项，如下图所示：



调试工具界面 8 – 属性变化断点调试

在 HTML 视图对监听元素进行点击操作（在 JavaScript 脚本中对该元素添加了点击监听），浏览器会自动跳转到 JavaScript 代码中，如下图所示：

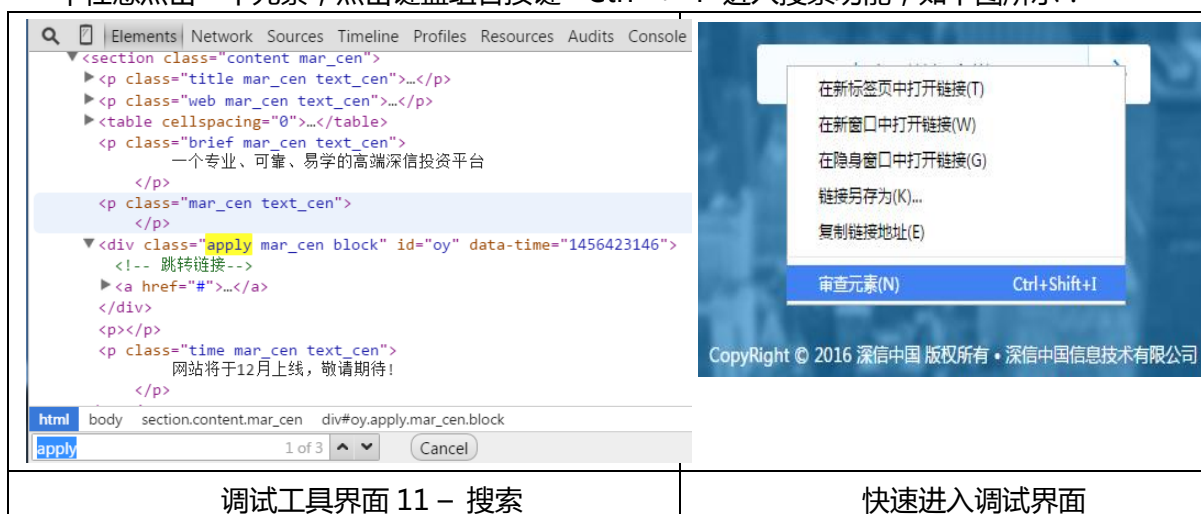


通过 B 区切换到 Properties 选项，可以看到选中元素中的各种信息属性，仅作为操控使用，如下图所示：



上图中的 B 区没有显示 Properties 选项，它被隐藏了，在 Event Listener 右边的扩展符号里，通过那里进入该选项。

如何在复杂页面的众多元素/标签中快速定位目标元素？方法 1. 在打开 Element 选项中任意点击一个元素，点击键盘组合按键 "Ctrl" + "F" 进入搜索功能，如下图所示：

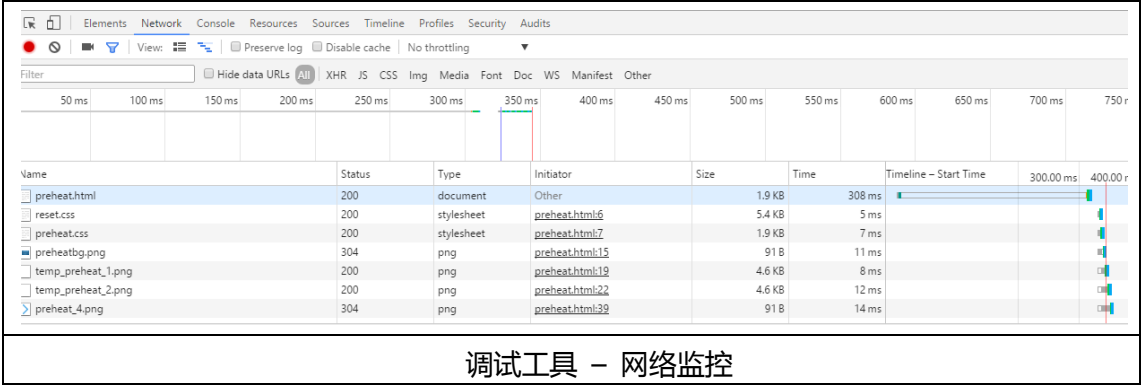


结果以高亮显示，如果未进入开发者工具，该组合按键将会启用页面内容搜索功能。第二种方法，通过鼠标选择某个元素，单击右键，选择 "审查元素"（不同浏览器使用方法不一致），浏览器将会自动进入开发者工具并定位到 DOM 树中的那个元素上，以上就是

Element 选项中经常使用到的功能模块。

1.3.2 知识点 2 网络监控

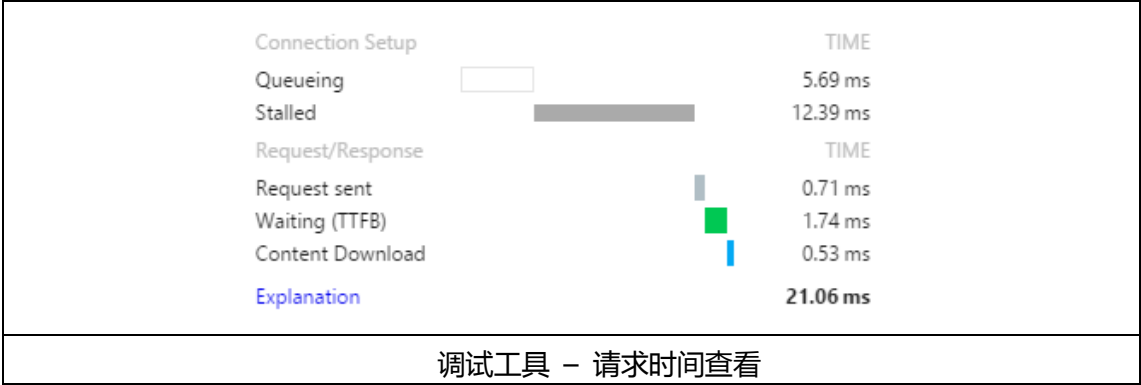
Network 是一个监控当前网页所有 HTTP 请求的面板，它的主体部分展示的是每个 HTTP 请求，每个字段（列）表示着该请求的不同属性和状态。



其中下表用来说明上图中每个字段（列）的用途。

字段名	说明
Name	请求文件名称
State	请求完成的状态，上图中 200=请求成功，304=缓存，具体的 code，请参考 http 协议的状态码
Type	请求类型
Initiator	请求源，该请求通过某个 js 文件发送
Size	下载文件大小或请求占的资源大小
Time	请求或下载时间
TimeLine	该请求在发送过程中的时间轴状态（开始下载时间，等待加载时间，自身下载耗时）

在上表中最后一个 TimeLine 字段说明中，通过鼠标移动到该字段下的时间轴上可以查看这些时间轴的更详细信息，如下图所示。



点击面板中的任意一条 HTTP 信息，会在底部弹出一个新的面板，其中该面板记录了该条 HTTP 请求的详细参数 header（表头信息、响应信息、请求信息）、preview（返回的格式化后的文本信息）、response（响应信息内容的数据内容）、cookies（返回或请求所带的 cookies）、Timing（请求过程中时间变化），通过这些信息可以得知请求的详细内容。

其中需要说明的是 cookie 信息，在单机状态下默认是不会显示 cookies 项的，因为

cookie 是由服务端返回的 ,客户端接收到该 cookie 后保存起来 ,第二次请求带上该 cookie 验证该身份 , 在后面的章节中将会提到如何设置 cookie。

最后在主面板的底部记录了整体网络请求状态的基本信息 , 如下图所示 :

0 / 227 requests | 0 B / 805 KB transferred | Finish: 50.72 s | DOMContentLoaded: 1.90 s | Load: 22.14 s

调试工具 - 基本信息

1.3.3 知识点 3 了解 Resources

Chrome 新版本中资源在 Application 标签选项下 , Resources 主要向我们展示了本界面所加载的资源列表、cookie 信息、本地存储数据 , 缓存数据等一系列信息 , 在该界面中可以对页面的这些数据进行修改、删除等操作。

414 x 736

Application

Manifest

Service Workers

Clear storage

Storage

Local Storage

Session Storage

IndexedDB

Web SQL

Cookies

http://www.163.com

http://g.163.com

http://pos.baidu.com

https://pos.baidu.com

https://eclick.baidu.com

http://popme.163.com

http://s.csbew.com

Name	Value	Do...	Path	Ex...	Size	HT...
City	0755	.16...	/	20...	8	
NTEs_hp_textlink1	old	.16...	/	20...	20	
Province	020	.16...	/	20...	11	
UM_distinctid	15ab82cb1f34e-0eea1142f8...	.16...	/	20...	72	
_gads	ID=aee851e46d66a4a0:T=14...	.16...	/	20...	75	
_ntes_nnid	d4c442422f39500b2fdcd604...	.16...	/	21...	56	
_ntes_nuid	d4c442422f39500b2fdcd604...	.16...	/	20...	42	
afpCT	1	.g.1...	/	20...	6	
vinfo_n_f_n3	744282991b1147ec.13.1489...	.16...	/	20...	76	
vjlast	1489148605.1489538907.11	.16...	/	20...	30	
vjuids	79b7b9f73.15ab82c7373.0.f9...	.16...	/	20...	43	

调试工具 - 网站资源信息

该选项中的左侧列表中选择 Frames 记录着网页所要用到的静态资源 , 比如图片、样式表文件等。如用户访问该网页 , 看到某个图片适合用作为素材 , 想拿到该图片源文件但无法通过鼠标右键保存图片 , 可通过打开该选项 , 展开 Frames 选项 , 通常在 images 目录下查找目标文件 , 找到图片后可以下载该图片文件。鼠标右键有多种方法可以选择 , 如下图所示 :

Elements Console Sources Network Timeline Profiles Application Security Audits

Cookies

Cache

Cache Storage

Application Cache

Frames

top

about:blank

r

r

Images

027010MUlogo0313.jr


060fa59b47754ae5b9l

20091021173129385b

20091204161709a5b4

20091204180357acflf

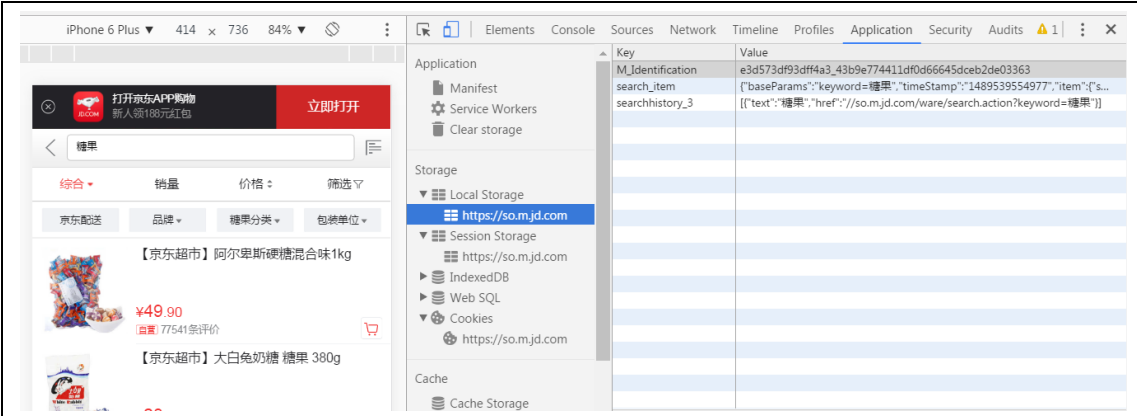
51.6 KB 600 x 300 image/jpeg



Copy image URL
Copy image as Data URI
Open image in new tab
Save...

local storage 和 session storage 这 2 个选项属于本地离线存储技术,它们是 HTML5 新的规范。接下来要通过一些 JavaScript 代码结合 HTML5 中的本地存储技术,主要讲解如何使用以及如何通过该工具进行调试。

比如"京东"网站,通过开发者工具的该选项可以看到以下离线存储的数据。



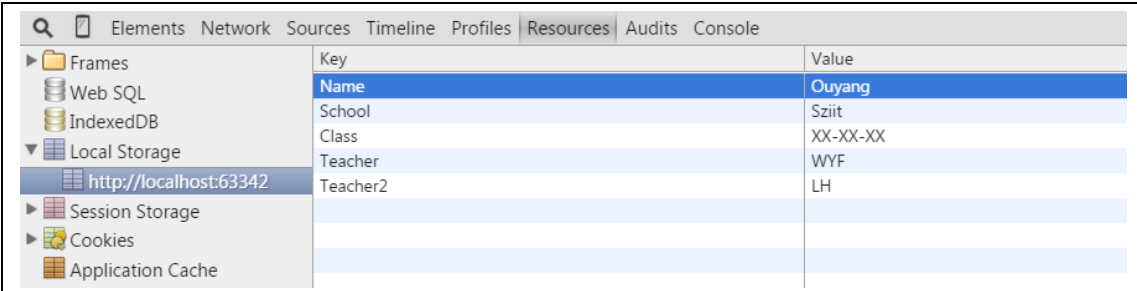
调试工具 - 网站离线存储信息

如何在 HTML 中加载 JavaScript 代码,前面的单元已有详细讲解,但需要注意的是加载的顺序问题,一般需要对 DOM 进行编程的 JavaScript 脚本应当在整个 HTML 页面加载完毕后才开始执行,如何办到呢?除了调整脚本文件加载顺序,还可以通过 window.onload 方法,该方法意思指页面加载完毕时,执行该方法(这个方法在后续项目开发中有重要作用)。本节通过第一种方法,调整加载的位置,把加载 JavaScript 脚本的操作放在末尾标签中。随着硬件速度的提升为开发便利常常把 JS 脚本放在 Head 中,但是当脚本数量很多时如果首先加载 js 文件就不能同时加载<body>部分,页面可能需要很长的初始化时间,其他元素(如图像)得不到第一时间的加载,这样会影响到用户体验。

...省略

```
<script src="xxx" type="text/javascript"><script>
</body>
</html>
```

由前面内容讲解,可对当前网页的本地存储数据进行操作,通过以下图形化操作,存储几个数据,然后代码显示出来,操作如下:



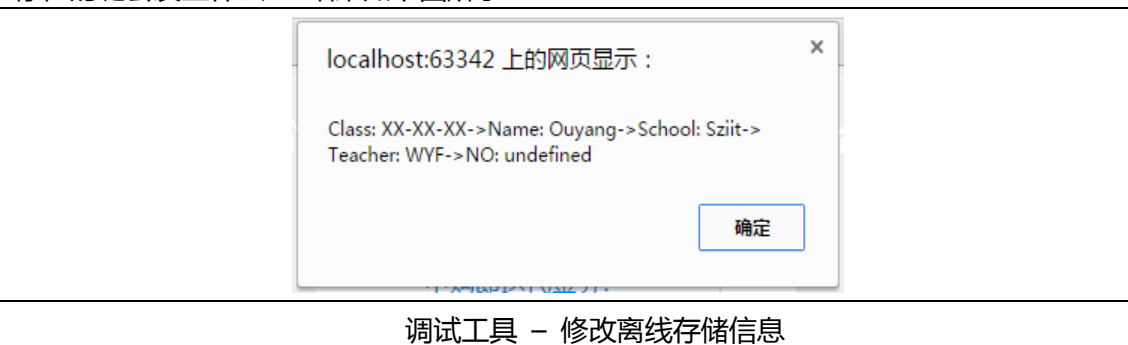
调试工具 - 修改离线存储信息

Web 的数据存储结构是一个键值对,一个键 (key) 对应一个值 (value),其中键和

值是成对出现的，通过键可以查询到值，反过来却不行。键不可以有重复，值可以重复，键一旦重复，后面添加的键会覆盖相同的键。在 JavaScript 代码中显示出 local Storage 的数据需要调用 `alert()` 方法，该方法作用是使浏览器弹出一个“框”，“框”可以显示信息也可以作为交互的手段，代码如下：

```
<!--...省略网页内容代码-->
alert(
"Class: " + localStorage.Class +
"->Name: " +   localStorage.Name +
"->School: " + localStorage.School +
"->Teacher: " + localStorage.Teacher +
"->NO: " + localStorage.no //错误示范
);
```

上图中的代码，向 `localStorage` 对象取出对应的数据，前文提到它的数据存储结构是一对键值组成，想要获得值，首先必须知道键是什么。`localStorage.xx` 中的 `xx` 指的就是“键”名称，并且它的命名对大小写敏感。其中最后一行代码，向 `localStorage` 对象取一个不存在的键会发生什么？结果如下图所示：



前文是直接通过开发者工具中 Resources 选项下的 `localStorage` 子项对本地数据进行修改。那在正式环境下如何使用 JavaScript 对数据进行存储呢？

前文中显示本地数据的调用方法是 `localStorage.xx` 的方式，对 `localStorage` 的存储方式也采用同样的方法 以 `localStorage.XX = "XX"` 为例，如下图代码所示

```
localStorage.MyFriend = "ZJW";
```

上图代码中使用 `localStorage` 接口向本地存储写入了一个 `MyFriend` 的键 (`key`) 和一个“ZJW”的值 (`value`)。重新刷新页面，在 Resources 选项可以看到以下数据：

Elements Network Sources Timeline Profiles Resources Audits Console		
<div> <div>Frames</div> <div>Web SQL</div> <div>IndexedDB</div> <div>Local Storage</div> <div>http://localhost:63342</div> <div>Session Storage</div> <div>Cookies</div> <div>localhost</div> <div>Application Cache</div> </div>	Key	Value
	Class	xx-xx-xx
	MyFriend	ZJW
	Name	Ouyang
	School	Sziit
	Teacher	WYF
	Teacher2	LH

调试工具 – 修改离线存储信息

与之对应的 Session Storage 用法也和 Local Storage 相似，session Storage 不是一种持久化的本地存储，仅仅是会话级别的存储。而 localStorage 用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。

Cookies 选项存储的是所有 Cookie 值，那什么是 Cookie 呢？通俗讲它是由服务器用来标识客户端的创建的 ID，例如浏览器进行一次 HTTP 用户验证请求时，验证成功则服务器创建标识 ID（cookie）发送给客户端，当客户端第二次 HTTP 请求时带上 cookie 值，服务器可以分辨出客户端，从而做出不同的响应。

Cookie 值并不是像 local Storage 一样由自己手动处理并存储，它的存储过程由浏览器自身完成，其中需要后台服务器与之配合才可以。那它的 Cookie 值是如何存储的呢？如前文所讲默认是不会有 Cookie 值的，浏览器又怎么知道某次请求有 Cookie 值？这个就刚刚所讲需要后台服务器的配合才可以完成这项工作。

为了更好地理解 Cookie 的内容，需要借助 Network 选项进行辅助（前文提到可以通过它查看 HTTP 请求详细信息）。客户端（网页浏览器）第一次对服务端进行 http 请求时，服务端对其进行响应（Response）时为 headers（头部信息集合）添加 header（头部信息）项，header 的构成形式由一个键和一个值组成，如下图所示：

Name	Headers	Preview	Response	Cookies	Timing
preheat.html	200 OK				
reset.css					
preheat.css					
preheatbg.png					
jquery.js					
preheat.js					
temp_preheat_1.png					
temp_preheat_2.png					
preheat_4.png					
userLogic.do					
favicon.ico					
11 requests, 1.21 KB total					

调试工具 – 查看详细请求信息

添加的 header 选项的代码如下图所示：

```
//该代码是 node.js 部分代码
response.setHeader("Set-Cookie", "xx");
```

上图中的 "Set-Cookie" 为固定写法，后面的值不固定，由服务器创建，用于辨别客户端。浏览器接收到该请求后，如发现返回的 HTTP 请求中的 header 部分有"Set-Cookie"选项，此时浏览器会自动将它所对应的值保存下来，可以通过 JavaScript 访问该浏览器 cookie 的值，但该值是只读的。也就说只要返回部分有该标志，浏览器都会自行保存。

Elements Network Sources Timeline Profiles Resources Audits Console						
<ul style="list-style-type: none"> Frames Web SQL IndexedDB Local Storage Session Storage Cookies <ul style="list-style-type: none"> 127.0.0.1 Application Cache 	Name	Value	Domain	Path	Expire...	Size
	_sid	860e1fb60b9cdbad92195b03f439f...	127.0.0.1	/	2063~...	68
调试工具 – 查看 Cookies 信息						

上图中，除了 Name 和 value 之外，还有其余的选项，以下表所示分别介绍用途：

选项（列）	说明
Name	Cookie 名字
Value	Cookie 实际值，唯一
Domain	作用域
Path	路径
Expires/Max-Age	过期时间
Size	Cookie 大小

需要说明的是 Name 选项和服务端返回的"Set-Cookie"这个命名没有太大的关联，实际上，上表所有的值都是"Set-Cookie"对应的 Value 值集合，以上图 Cookie 代码所示：

...部分 Node.js 代码
<code>response.setHeader('Set-Cookie', '_sid=' + xx + ';path=/' + ';Max-Age=' + XX);</code>

如上图所示，每一个选项（列）都由 ";" 结束，浏览器才可以顺利的分割出每个子项（列）的值，在 Network 选项中也可以清楚的看到。

▼ Response Headers view source
Connection: keep-alive
Content-type: application/json; charset=utf-8; text/plain
Date: Mon, 25 Jul 2016 09:46:46 GMT
Set-Cookie: _sid=860e1fb60b9cdbad92195b03f439f28784188f600d2c11430acc11be3280944b;path=/;Max-Age=1469526406
Transfer-Encoding: chunked

当第一次请求完毕后，浏览器已经拿到了 Cookie 值，在第二次 http 请求时，我们应该如何为该 HTTP 请求携带上该 cookie 值？第一次请求的 HTTP 的 response 和 request。

The screenshot shows the Chrome DevTools interface. The top panel displays the 'Response Headers' for an HTTP response. The 'Set-Cookie' header is highlighted with a red box, showing the value: `_sid=860e1fb60b9cdbad92195b03f439f28784188f600d2c11430acc11be3280944b;path=/;Max-Age=1469526406`. Below this, the 'Request Headers' are visible. The bottom panel shows the 'Resources' tab with the 'Cookies' table expanded. The table has columns: Name, Value, Domain, Path, and Expires / Max-Age. The first row shows the cookie name '_sid' and its value '860e1fb60b9cdbad92195b03f439f28784188f600d2c11430acc11be3280944b'.

Name	Value	Domain	Path	Expires / Max-Age
_sid	860e1fb60b9cdbad92195b03f439f28784188f600d2c11430acc11be3280944b	127.0.0.1	/	2063-02-17T19:33:33...

调试工具 – 查看 Cookies 信息

从上图中可看到 Response 中的 headers 集合中有"Set-Cookie"值，并且在 Resources 选项中的 cookies 也有对应的 cookie 值。保持当前页面，进行第二次 HTTP 请求，观察 response 和 request 信息有什么变化，如下图所示：

The screenshot shows the Chrome DevTools interface for a second HTTP request. The 'Request Headers' section shows the 'Cookie' header with the value: `_sid=860e1fb60b9cdbad92195b03f439f28784188f600d2c11430acc11be3280944b`, which is highlighted with a red box. The 'Response Headers' section shows the 'Set-Cookie' header with the updated value: `_sid=860e1fb60b9cdbad92195b03f439f2872f7098bc3f5ccb3f873661ec9886c674;path=/;Max-Age=1469528074`, also highlighted with a red box. The 'Resources' panel on the left shows the request file 'userLogic.do'.

调试工具 – 查看 Cookies 信息

从上图中可以很明显发现第二次 HTTP 请求中 request header 携带上了 Cookie 值，并且它的 cookie 值和第一次是一样的，也就是说从拿到 cookie 值到上传 cookie 值等一系列操作都由浏览器自身完成了，在这一过程中唯一需要配合的就只有服务端了。

出于安全考虑，Cookie 值从保存到上传都由浏览器自动处理，不允许开发者进行管理，在早期的浏览器可以对 Cookie 值进行读/写操作。

补充说明：前文提到断点可通过鼠标单击文本编辑/查看区域的行号来完成，还有另外一种断点方法，通过在 JavaScript 代码文件里添加上 debugger 关键字即可，详细的调试方法在下一节介绍，如下图所示：

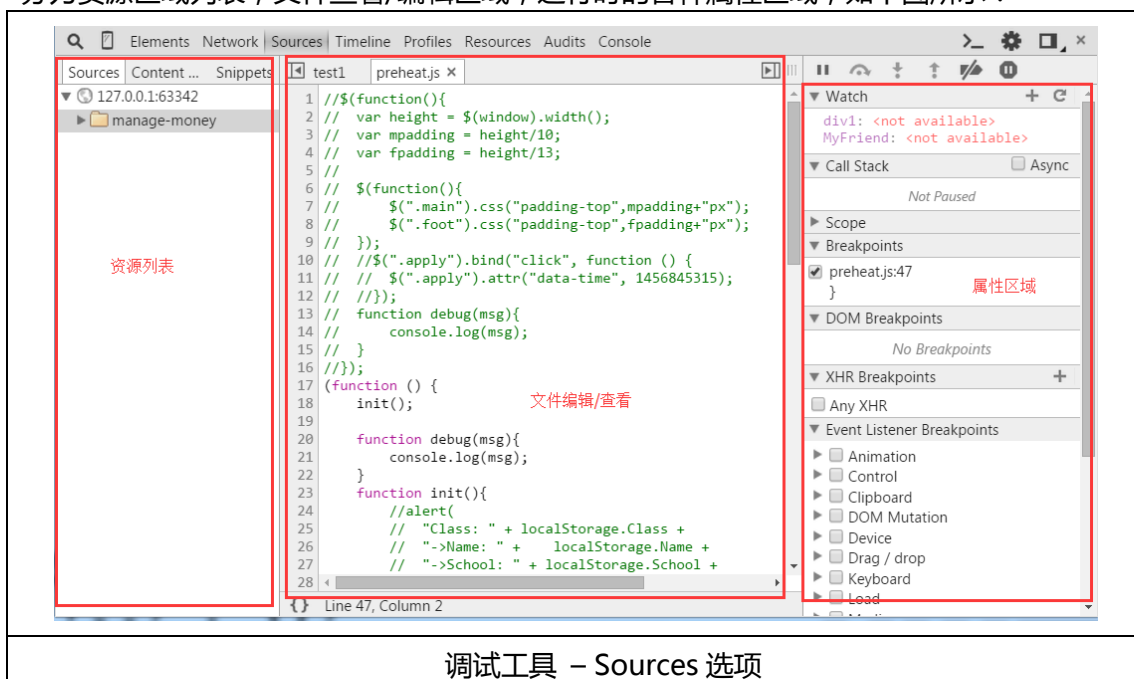


注：本项目使用 Node.js 作为后台支持，关于对 Node.js 具体讲解请参阅本书的姊妹篇《HTML5 开发实战》中内容。

1.3.4 知识点 4 调试 Sources

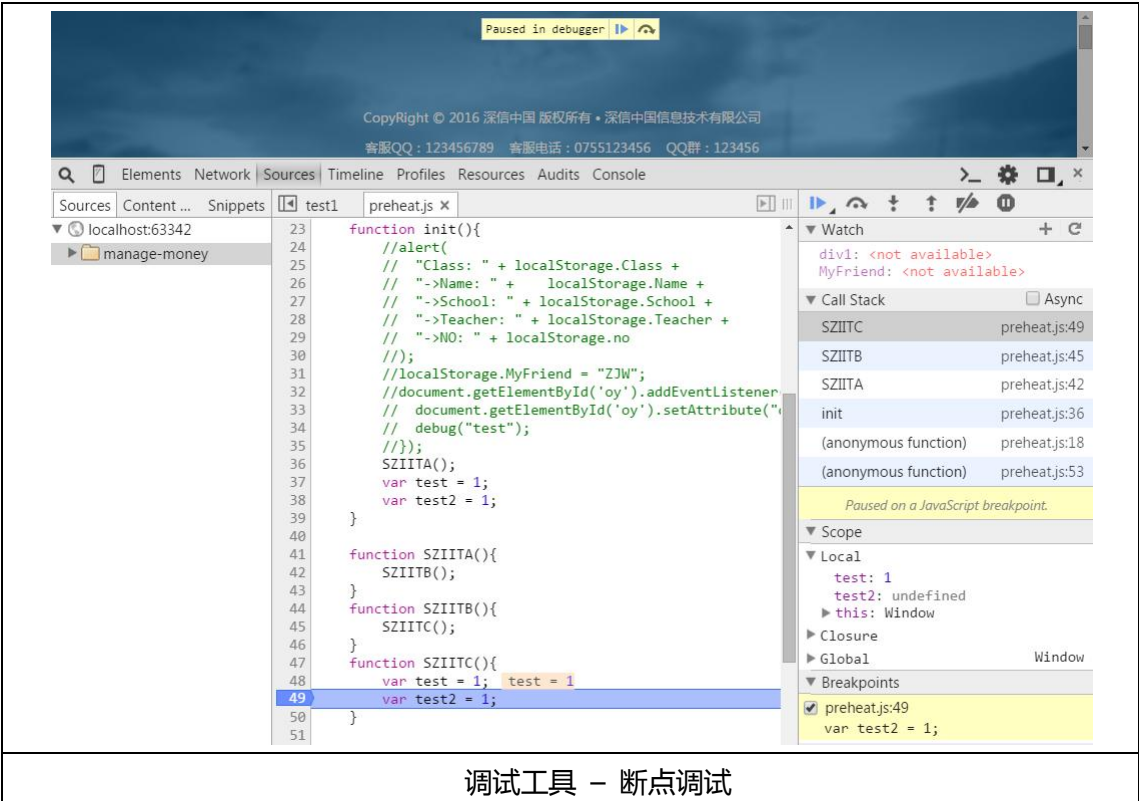
Sources 选项的内容主要是用来调试 JavaScript，可以像 IDE 一样对程序打断点，通过 step 一步一步查看程序流程以及观察每个变量值内容。这样就简化了以 alert 弹出窗口进行调试代码了。可结合第 5 单元《JavaScript 与 DOM 编程》内容一起学习。

Sources 选项中的面板可以分为 3 块互相关联的区域，互动共同实现一个重要的功能：监控 JavaScript 在执行期间的活动，通俗点说就是“断点”。Sources 选项中 3 块区域可以分为资源区域列表，文件查看/编辑区域，运行时的各种属性区域，如下图所示：



先看资源列表区域，它有 3 个 tab 页面，分别为 Sources，Content Script，Snippets。Sources 页面下主要显示当前网页加载文件，例如 css，js 等文件（它不包含 cookie 和图片）。双击一个文件，该文件内容会出现在文件编辑/查看区域中显示，如果选择的是

JavaScript 文件，可以在该区域中显示的代码行号上点击即设置了断点。当 JavaScript 文件在执行过程中执行到了该标记行，便会自动停止，并等待你的下一步命令。



从上图可以看到 JavaScript 执行到断点处各个区域的变化，首先是属性区域(右下角) 中 Breakpoints 记录信息会变高亮，然后是右侧的 Local 列出了各个变量信息以及指向的对象的详细信息，这就帮助我们直观的知道此时 JavaScript 的执行状态。同样，你可以把鼠标移动到某个变量上，它会弹出一个框，直接显示该变量的信息。

接之后可以选择按"F8"继续执行后面代码，但常常是以步进的方式按"F10"继续按照 JavaScript 的程序流程一步步地走下去，在执行到某个函数时，按"F11"可以进入到该函数体内观察代码的执行流程和变量内容，也可以通过界面上的选项进这些操作，推荐使用快捷键操作。



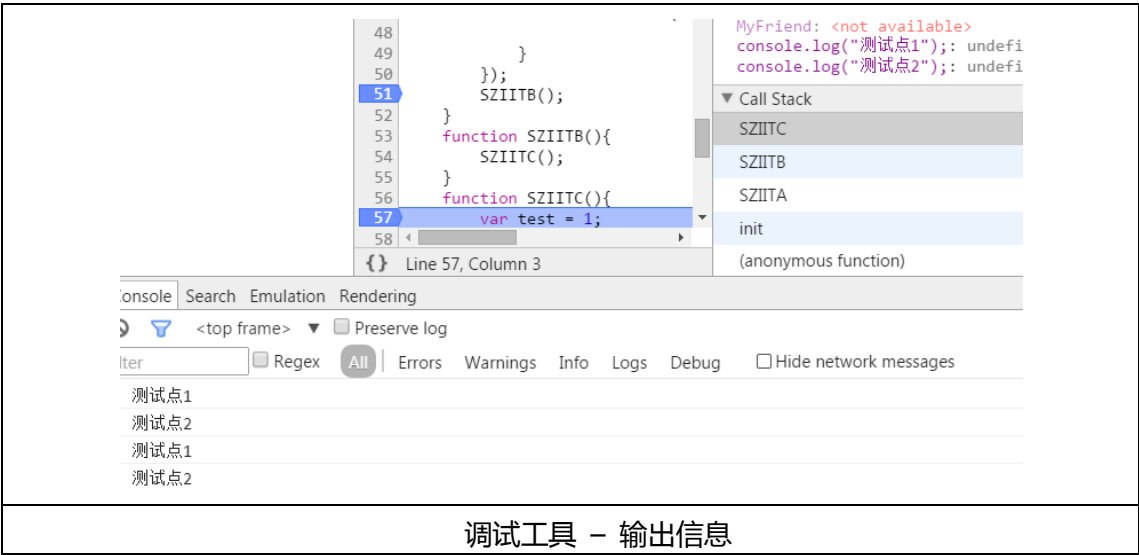
- 上图中红色字体的数字，它们分别代表：
- 1、执行到下一个断点，如果没有则执行到完所有程序流程 (F8)；
 - 2、不跳入函数中，继续执行下一行代码 (F10)；
 - 3、跳入函数 (F11)；
 - 4、从执行函数中跳出；
 - 5、禁用所有断点，不做调试；
 - 6、程序运行异常时中断开关；

接下来看各属性区域，从第一个"Watch"选项开始，它的作用是为目前断点添加"表达式",使得每次断点往下一步都会执行你写下的表达式 ,需要注意的是这个功能请谨慎使用，可能会导致你写下的表达式重复执行多次。



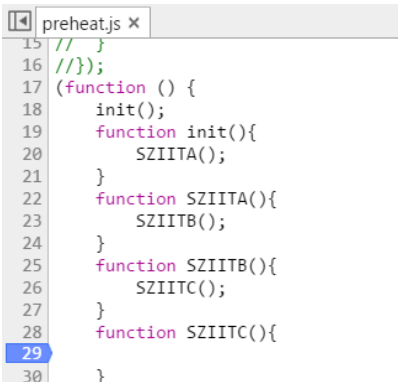
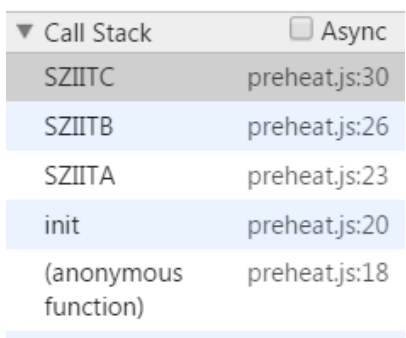
调试工具 - 查看断点信息

上图中后面的 console.log(); 是后面添加的，每次 js 文件执行到断点处就会执行该代码，添加的断点每次都会执行，比如下图中定义了 2 个断点， Console（下节将讲解）选项将会输出 2 次信息。



调试工具 - 输出信息

接下来是"Call Stack", 中文意思是调用栈 ,它的作用是显示断点处 ,方法的调用情况，比如上图中的 Call Stack 明确的显示出断点 57 行中，各个方法之间的调用情况。

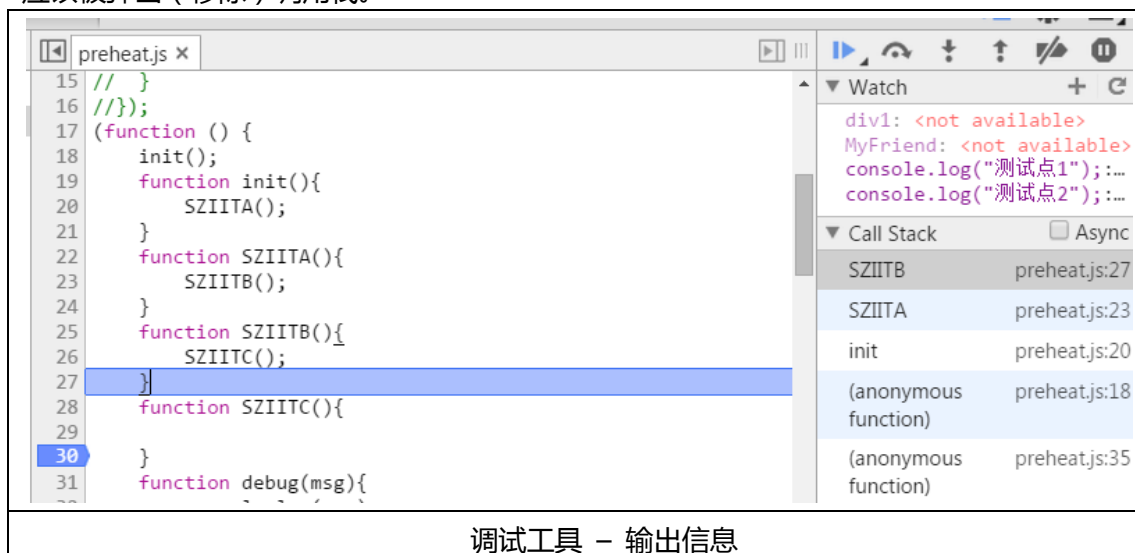
	
调试工具 - 输出代码信息	调试工具 - 输出栈信息

从调试栈信息图可以清晰的明白调用关系是 init()->SZIITA()->SZIITB()->SZIITC() ,自然在 Call Stack 中按出栈顺序就是 C->B->A->init()。

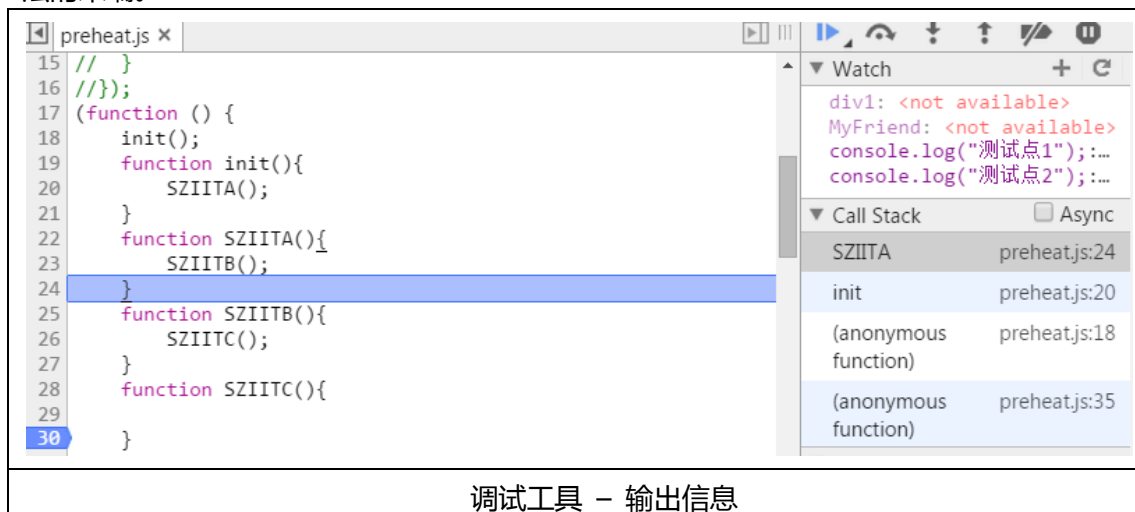
上图中可看到它们的调用关系反了过来 ,那为什么会这样显示？"调用栈"中栈是计算机里常见的一种数据结构 ,作为一个线性表 ,它被限制仅允许在表的一端进行插入/删除运算。

插入数据端称为栈顶，把另外一端称为栈底，向栈插入一个新元素又称为进栈、入栈或压栈 (push)，它总是把最后插入的元素放在栈顶。从一个栈删除元素又称为出栈或退栈(pop)，把栈顶元素删除后，使其相邻的元素称为新的栈顶元素。

作为只能在一端进行插入/删除操作的表，它按照先进后出的原则存储数据，先进入的元素在栈底，最后一个进入的元素在栈顶如上图所示，执行完 SZIITC()方法后，该方法结束应该被弹出（移除）调用栈。



蓝色行高亮状态指的是执行到了 SZIITB()方法的末端，继续按"F10"执行到 SZIITA()方法的末端。

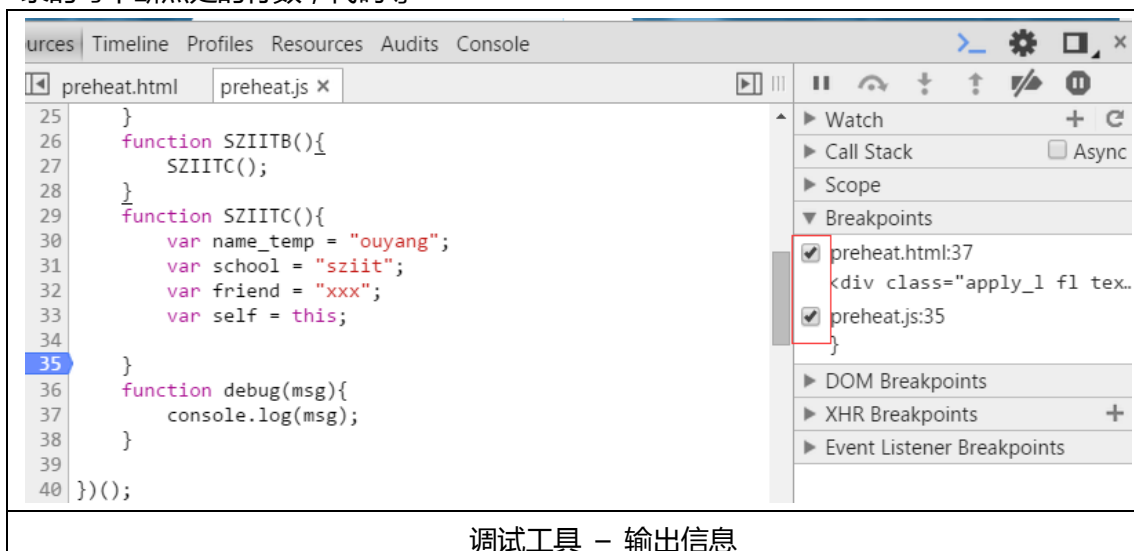


继续执行去可看到栈的操作是如何将方法元素入栈和出栈。下一个是 "Scope"选项，它可以查看当前断点出内的的变量值，当前也可以文本编辑/查看区域浏览，不同的是该选项，可以看到更多的变量，比如作用域，闭包，全局变量等。

当在函数 b 中访问一个变量的时候，搜索顺序是先搜索自身的活动对象，如果存在则返回，如果不存在将继续搜索函数 a(引用的函数)的活动对象，依次查找，直到找到为止。如果整个作用域链上都无法找到，则返回 undefined。如果函数 b 存在 prototype 原型对象，则在查找完自身的活动对象后先查找自身的原型对象，再继续查找。这就是 Javascript 中的变量查找机制。



下一个是 "Breakpoints" 选项，从它的命名就可以知道它的实际作用，断点信息，它记录的每个断点处的行数，代码等



注意到上图中该选项下的选项框了吗，如果把默认选项的 "√" 去掉，则其对应的断点在调试时就不会继续执行。

下一个是 "DOM Breakpoints" 选项，监听 DOM 树中元素的变化，比如属性，节点，事件状态等，在前面的 Element 选项讲解过关于该知识点的内容，本节就不在重复一次了。

再下一个是 "XHR (XMLHttpRequest) Breakpoints" 选项，监听 Ajax 请求，它不需要开发者手动指定断点行数，通过点击它选项右边的 "+" 输入需要监听的 URL，当有该 URL 请求时，触发该断点，它会停在 `request.send()` 地方中断。



调试工具 - 监听 HTTP 请求断点

图中使用了 JQuery 插件，所以调试时它会跳入 JQuery 内部封装好的 Ajax 方法中。

选择 Any XHR 它将监听所有 Ajax 请求。在某些版本的 Chrome 开发者工具中该选项会出现小问题：即使取消还是会进入调试模式，可更换浏览器版本试试。

下一个是 "Event" Listener Breakpoints" 选项，它和上面一个选项用法类似，监听响应事件，它不能手动指定，只可以选择它定义好的类型，展开选项后里面的类型足够开发者使用了，比如 Mouse 选项中的 click 事件。



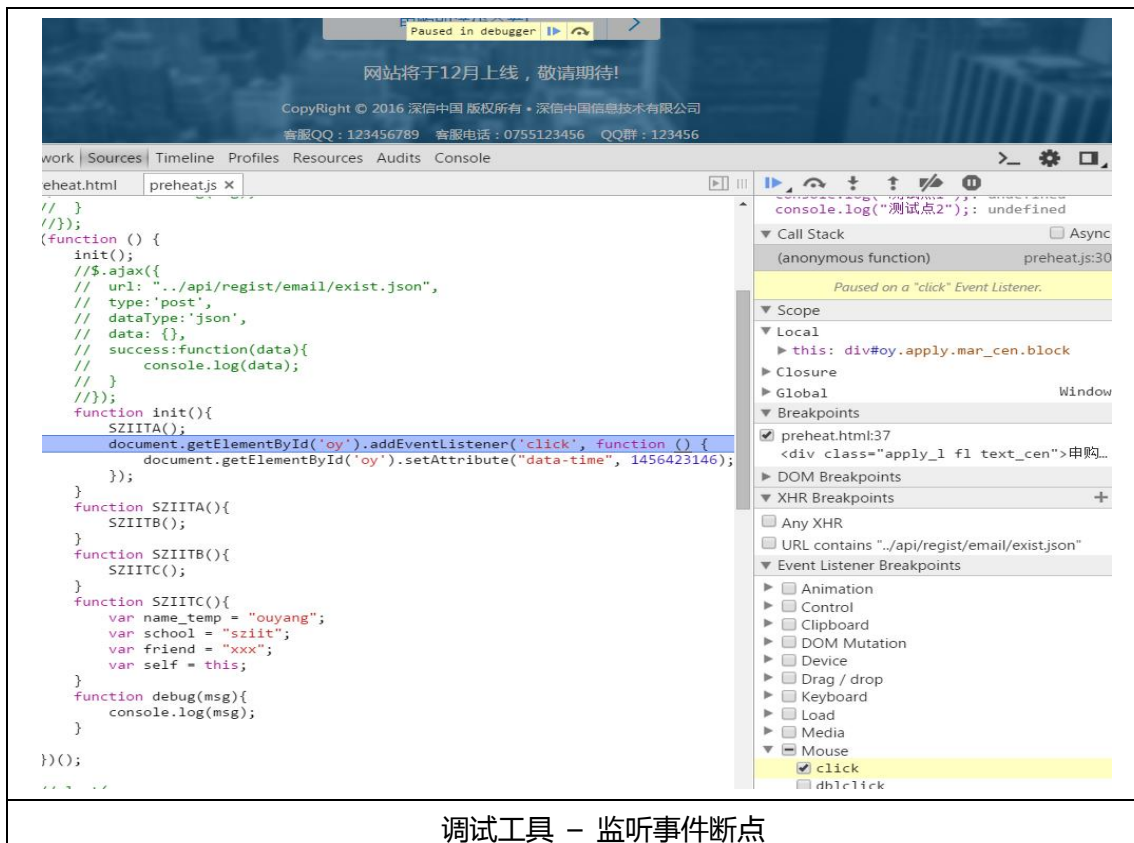
调试工具 - 监听事件断点

上图中 Mouse (鼠标) 选项下有很多子项，其中 click 事件是开发中使用频率相当高的事件。我们跟踪事件，如图中没有任何断点，只有一个 click 监听事件。



调试工具 - 监听事件断点

当点击网页上含有 click 监听事件的元素，该工具将会自动捕获该事件，并自动进入调试模式定位到该元素对应的 click 事件代码中，如下图所示：



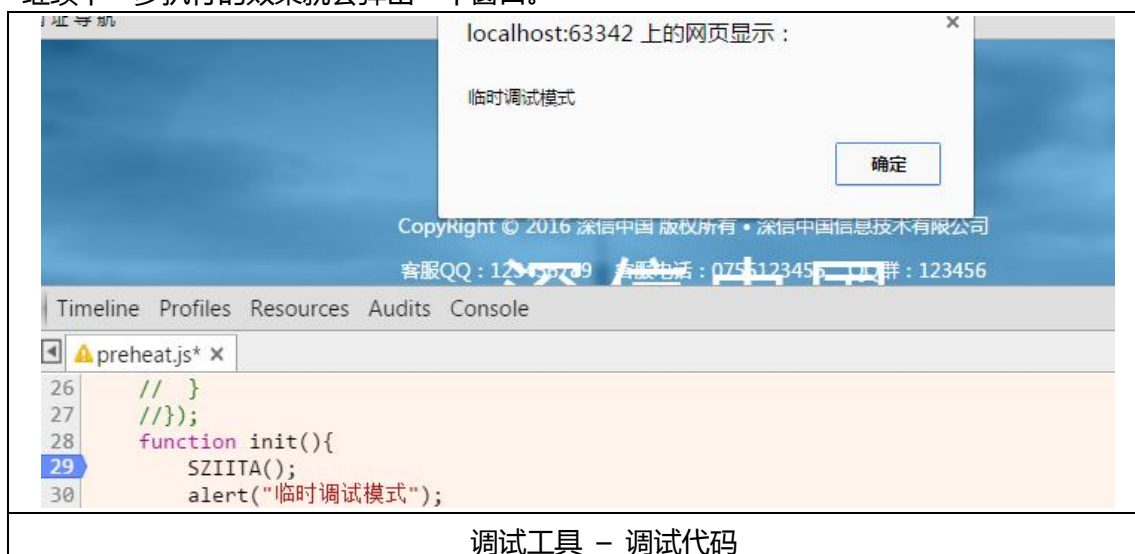
通过属性区域中的几个选项（比如 DOM/XHR/Event Breakpoints）可以方便的定位交互（http，手势）代码，大家可以尝试跟踪感兴趣网站中的网页使用，通过工具方法进行学习。

当你想要对当前网页的业务逻辑进行调试，可以采用 IDE 辅以网页工具，频繁切换于多个窗口运行页面效果容易出错。Sources 项的文本编辑/查看区域提供一种方法，可以对 JavaScript 文件进行小范围调试，不影响源文件，效果只存在当前浏览器中，它省去了频繁切换过程。以下是未做修改的样子。

调试工具 - 调试代码	调试工具 - 增加条黑色代码

再进行断点调试 通过文本编辑/查看区域对当前的js 文件进行修改 添加上调试代码，

添加并保存后的代码会使文本编辑/查看区域背景颜色改变,添加后的代码也可以立即生效,继续下一步执行的效果就会弹出一个窗口。

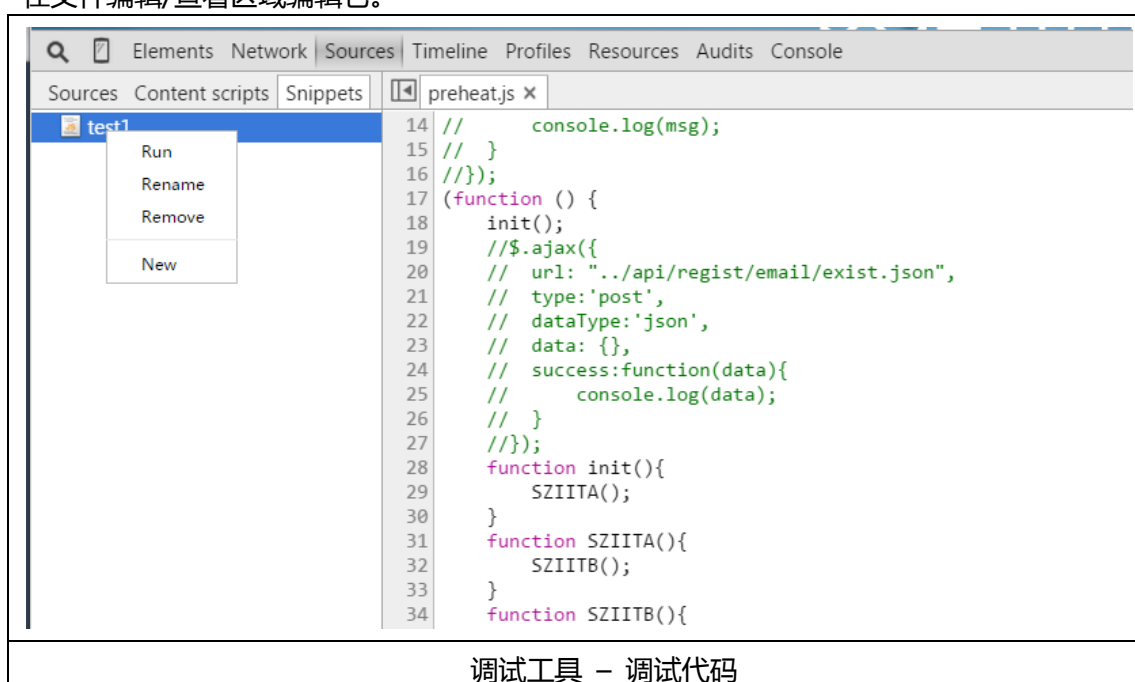


调试工具 - 调试代码



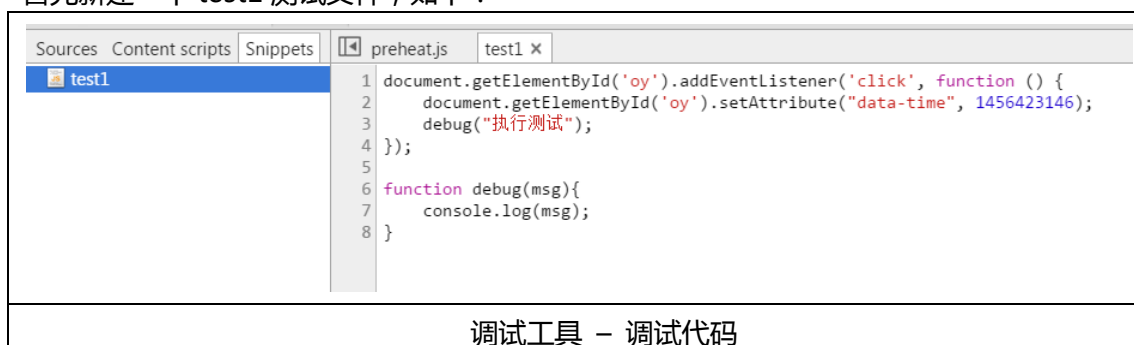
该方法只适用于小范围调试,并且代码只能应用一次,刷新后js文件恢复源代码。前文提到 Sources 面板下的 3 个选项加上图片描述,其中资源列表下还有 3 个子项, Sources 选项已经介绍过了, Content scripts 选项里包含着一些第三方插件或浏览器自身的js代码,实际开发中能用到的地方很少,可以更多的关注 Snippets 选项,它可以编辑js代码片段,这些片段相当于js文件一样可以存储在浏览器当中,即使重新关闭/打开也不会消失,也不会执行,除非是手动执行。它的主要作用可以使开发者编辑一些项目的测试代码时提供便捷,当你做完一个功能模块,可能需要写一些代码来测试模块正确性,在发布时你必须为它们添加注释符号或者直接删除它们,但在 Chrome 上就不用这么繁琐了。

在 Snippets 选项的空白处右键选项弹出菜单的"new"选项,建立一个新的文件,然后在文件编辑/查看区域编辑它。

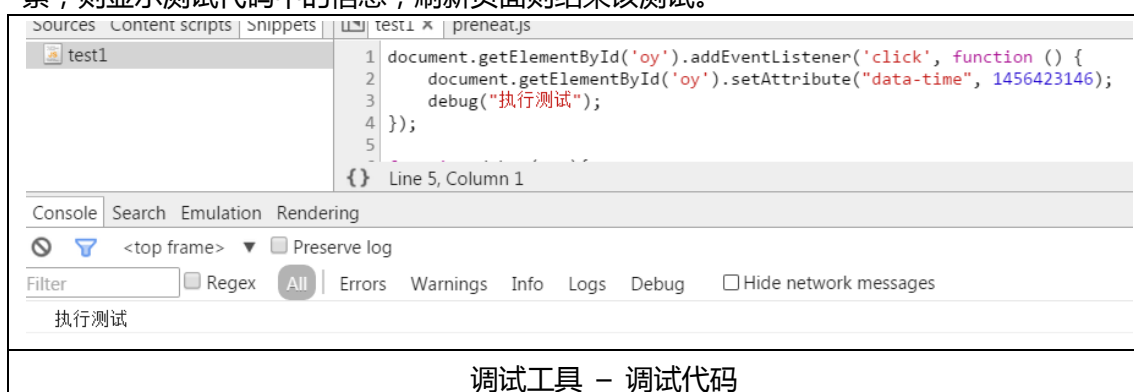


调试工具 - 调试代码

比如当前页面中需要为某个元素添加事件测试一些数据,但该代码并不用于正式环境中,首先新建一个 test1 测试文件,如下:



通过右键选中文件,点击 "Run" 选项,该测试文件被添加到当前页面中。点击测试元素,则显示测试代码中的信息,刷新页面则结束该测试。

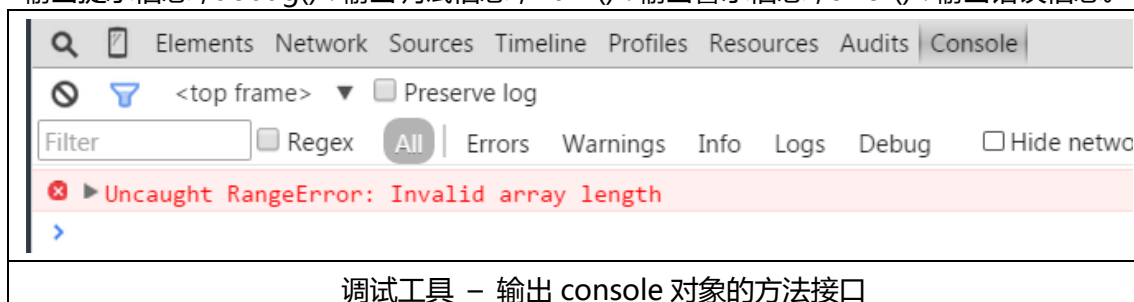


Snippets 的功能非常强大,可以直接操作 DOM 对象,例如可以使用它来调试某片段代码进行单元测试。

Sources 是开发者工具中最常用到的功能面板之一,它里面的许多功能对于前端工程师来说是非常有帮助的,不推荐在源代码里写调试信息,这样使得代码臃肿,后续修改变得繁琐。Chrome 开发者工具为开发者提供了强大的功能,应该好好利用。

1.3.5 知识点 5 Console 输出

Console 选项面板主要功能是输出浏览器异常/错误信息,当前 js 代码对 API 进行操作遇到错误便会输出对应的信息,比如数组越界等错误。程序逻辑错误可以通过 js 代码调用接口的方法输出调试信息,对象接口为 console(注意大小写),它有以下几个方法,info(): 输出提示信息,debug(): 输出调试信息,warn(): 输出警示信息,error(): 输出错误信息。



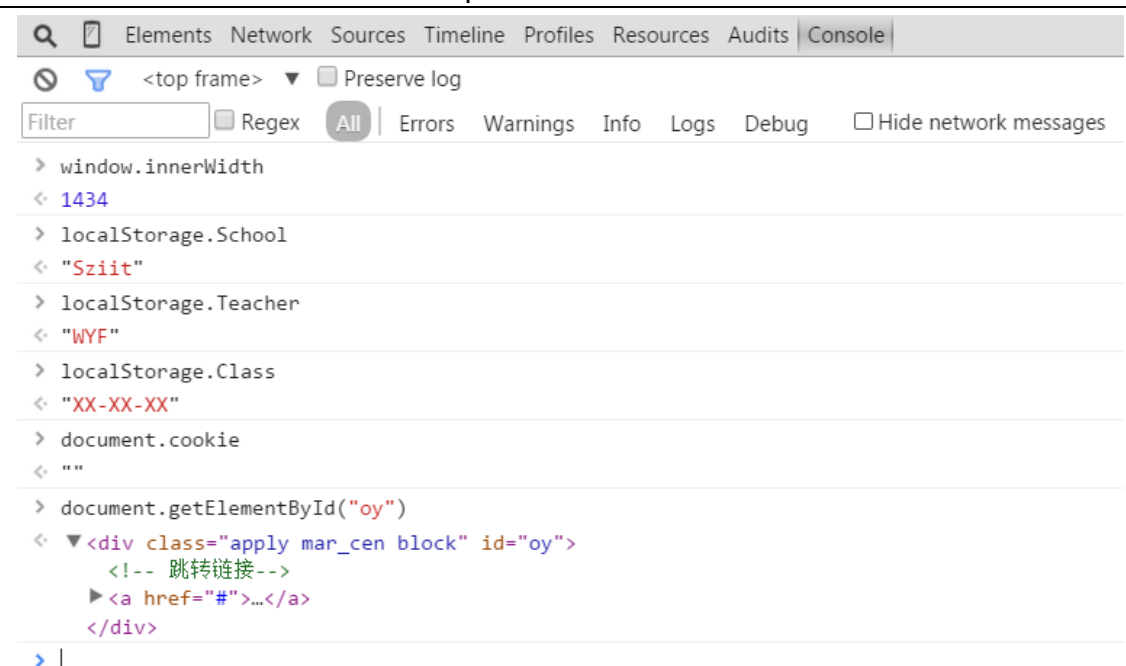
console.log()是 printf 风格,因此可以这么做,console.log("%s is %d years old",

"OY", 22);

```
> console.log("%s is %d years old", "OY", 22)
OY is 22 years old
```

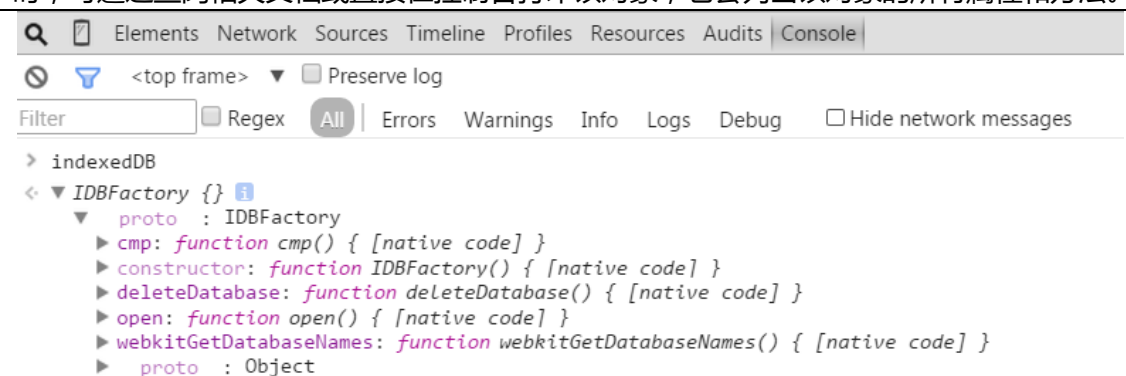
调试工具 – 输出 console 对象的方法接口

在 Console 面板的命令行中直接输入该代码，就可以看到效果，而不是前文提到的需要使用 JavaScript 代码来调用 console 接口来实现。该控制台本身可以通过一些命令来输出信息，但这些命令必须是浏览器中自带的，比如 window 对象、本地存储对象、console 接口、document 等，对于 JavaScript 自定义的对象和方法，该控制台无法调用。



调试工具 – 输出 html 代码和字符串

通过该面板中的命令行可以方便的进行简单的 js 调试。在不清楚某个浏览器内置对象时，可通过查阅相关文档或直接在控制台打印该对象，它会列出该对象的所有属性和方法。



调试工具 – 输出属性和方法

回到前文提到的 console.log()方法，还可以使用 "%c"模式，把第二个变量作为样式格式参数，例如：

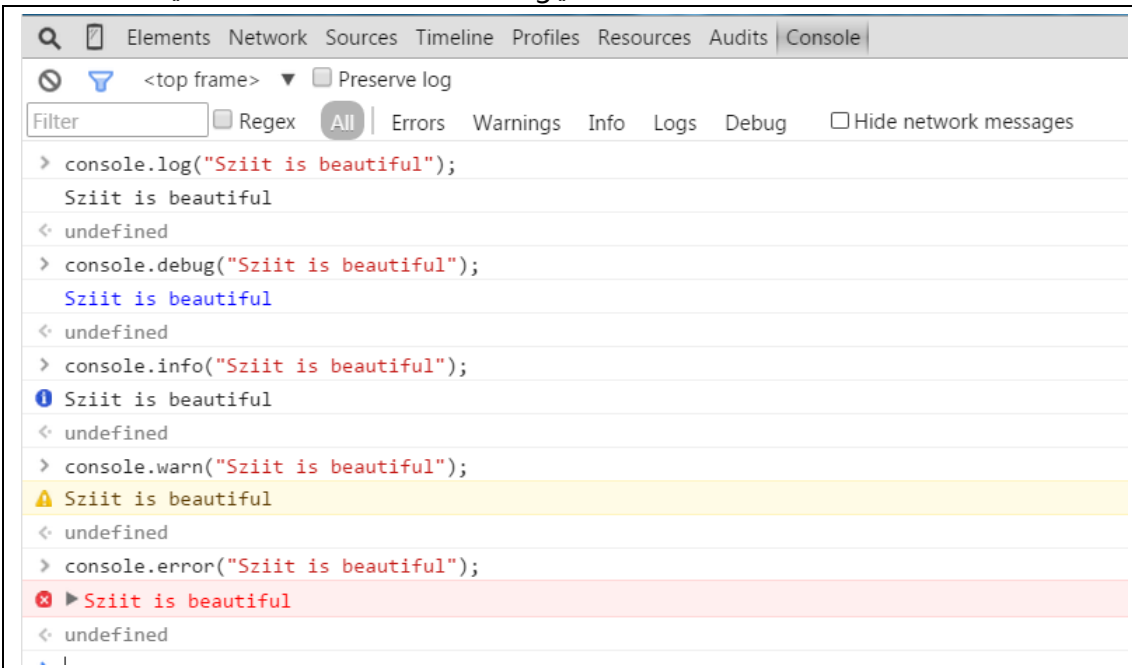
```
console.log("%c ShenZhen information university is beautiful.", "color: green; background-color: yellow");
```

log 默认的风格就会改变，变成下图的样子：

```
console.log("%c ShenZhen information university is beautiful.", "color: green; background-color:yellow");  
ShenZhen information university is beautiful.
```

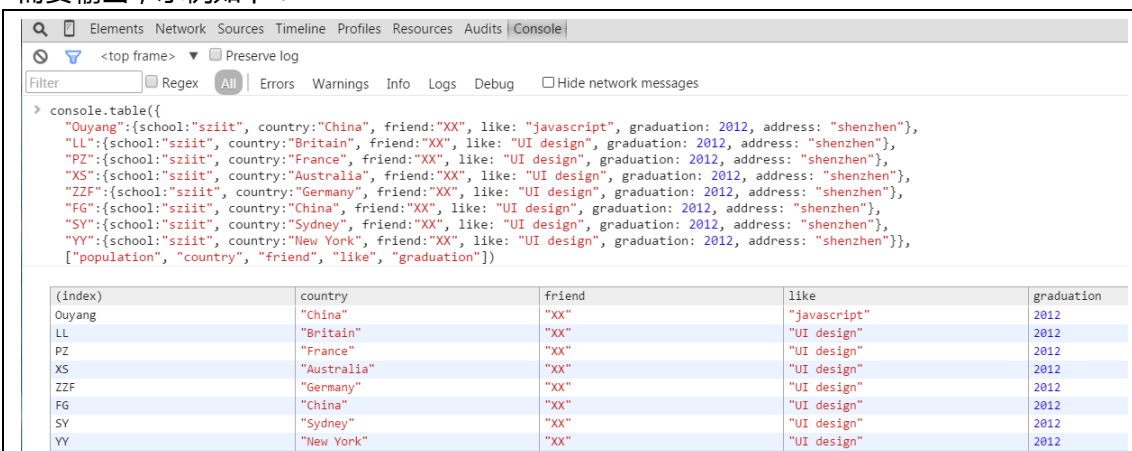
调试工具 – 改变输出样式

从目前使用情况来看，在 JavaScript 代码中使用 console 来调试程序，不推荐使用 debug()方法，在 Chrome 浏览器中，info()显示蓝色感叹号图标，warn()显示黄色感叹号图标，error()显示红色感叹号图标，debug()不显示图标。



调试工具 – 改变输出样式

除了以上常规信息输出外，console 同样可以处理输出表格数据，使用 console.table()方法，有两个参数，第一个参数是表格对象，第二个参数是可选变量允许开发者指定哪一列需要输出，示例如下：



调试工具 – 改变输出方法

console.assert() 用于测试表达式的真假，它有两个参数，表达式和显示信息，如果表达式是 "False"，信息会显示。

```
console.assert(!true, 'This is not true'); //输出 This is not true
console.assert(false, 'This is not true'); //不输出
```

Chrome 开发者工具中的 Console 面板功能强大，目前介绍的知识点是目前开发经常用到的。另外请打开百度首页，进入调试模式后查看 console，看看有什么内容呢？

1.4 任务实施

请使用 Chrome 调试工具测试本人或小组前面完成的工作，并验证下面三个问题的解决方案。

1.4.1 任务 1 查看网页资源



请通过 Elements 模拟调试，比如查看所在学校网站的样式设计、网页设计、布局设计，抓取感兴趣网站的静态资源。现在我们必须先回答一个问题：为什么把样式表放在头上？

无论是 HTML 还是 XHTML 还是 CSS 都是解释型的语言，而非编译型的，所以将 CSS 放在网页上方的话，在浏览器解析结构的同时对页面进行渲染。用户不用一直等待一个白屏，而是可以先看已经下载的内容。如果将样式表放在底部，浏览器会拒绝渲染已经下载的网页，因为大多数浏览器在实现时都努力避免重绘，这样可以避免页面结构光秃秃的先出来，之后 CSS 渲染页面又突然华丽起来，这样的页面浏览体验太“雷”了。

我们发现把样式表移到 HEAD 部分可以提高界面加载速度，这使得页面元素可以顺序显示。HTML 规范明确要求样式表被定义在 HEAD 中，因此，为避免空白屏幕或闪烁问题，最好的办法是遵循 HTML 规范，把样式表放在 HEAD 中。

1.4.2 任务 2 查看网络资源

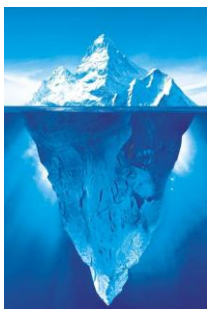


通过 Network 查看和网站交互的网络请求信息，分析请求类型、响应数据、抓包、数据信息体。现在我们又面临一个问题：JavaScript 和 CSS 应该包括在外部文件，还是在页面文件中？为什么？

在现实世界中，使用外部文件会加快页面显示速度，因为使用外部 Javascript 和 CSS 文件可以使这些文件被浏览器缓存，从而在不同的请求内容之间重用。如果内置 JavaScript 和 CSS 在页面中虽然会减少 HTTP 请求次数，但增大了页面的大小。

另外一方面，使用外部文件被浏览器缓存，则页面大小会减小，同时又不增加 HTTP 请求次数。请对这个问题使用浏览器中相关工具进行测试分析。

1.4.3 任务 3 查看调试信息



掌握使用 Console 输出调试信息,练习通过 window.height 等查看对象属性的方法。通过 Javascript 访问 DOM 元素没有我们想象中快,元素多的网页尤其慢,DOM 多少算多呢?

网页中元素过多对网页的加载和脚本的执行都是沉重的负担,500 个元素和 5000 个元素在加载速度上会有很大差别。想知道你的网页中有多少元素,通过在浏览器中的一条简单命令就可以算出:

`document.getElementsByTagName('*').length`,请试试如何在浏览器的地址栏上直接运行上面的函数,看看你喜欢的网站哪个 DOM 元素最多,响应又如何。

内容再丰富的网站,如果慢到无法显示也是毫无价值的;SEO 做的再好的网站,如果搜索蜘蛛或爬虫爬不到也是白搭;UE 设计的再人性化的网站,如果用户交互响应慢到无法忍受也是空谈,以上三种前端网页优化方法后续可以逐步体验。

1.5 单元小结

Chrome 开发者工具有众多的功能选项,常用的如 Sources 选项面板等功能对于前端工程师来说非常有帮助。Element 选项面板对于静态页面的调试很有帮助,可以通过它分析其它网站的设计思路,查看 CSS 代码,学习设计风格。还可以对网页的不满意的地方进行微调,对比以前只有修改源文件后刷新页面才可以看到效果。

Network 选项面板对于学习 HTTP 协议,充分认识 HTTP 中的每个变量,理解每个不同请求所对应不同的 header,熟悉每个 header 对于浏览器的影响和作用都比较有益。比如本章提到的 "Set-Cookie"。日

Resources 选项面板,可以充分利用该工具,收集各种图片素材。Console 选项面板可通过浏览器提供的 API,直接做一些简单的网页调试工作,不必依赖复杂的 IDE 工具。

前端开发中因大量新的技术被采用,代码更新快,所以不推荐在源代码里写调试信息以免使开发变得繁琐,Chrome 开发者工具为开发者提供了强大的功能,应当好好利用。

1.6 单元练习

1. 请用调试工具查看、跟踪百度、京东、淘宝、网易等页面,有惊喜噢!

2. IE11 及以上版本的开发者选项同样功能强大,如 UI 响应和资源分析工具可以帮助找到资源占用过高的原因;仿真面板下的功能就是面向所有人的简单又实用的好功能。仿真面板下的内容非常简单,可以在纬度和经度里填入想要模拟的位置,仿真成 iPad 跳过网站广告,还可以切换 IE11 的文档模式来查看网页在旧的 IE 浏览器上的显示效果,相信深入了解后你会有更多的收获。