# 浏览器专题二期报告

本期研究主要依托于一期研究基础，继续深入挖掘锦帛浏览器同Chrome浏览器之间的差异及锦帛浏览器安全漏洞相关问题，主要包括：

1. 获取到的UA差异
2. cookie和session机制原理及处理差异
3. 对本地存储(localstorage,sessionstorage)的支持情况差异
4. 对Vuejs渲染性能的benchmark测试
5. Chrome自43版本至今，更新了哪些重要安全漏洞？哪些漏洞可在锦帛浏览器复现验证？
6. 针对浏览器某个漏洞，处理流程是否可固化成方法论？如何查找漏洞详细信息？如何跟踪该漏洞修复情况？如何在锦帛上验证？
7. UA差异

UA：User Agent，“用户代理”，让服务器能够识别客户使用的操作系统及版本、CPU 类型、浏览器及版本、浏览器渲染引擎、浏览器语言、浏览器插件等。

* 1. 锦帛浏览器本地目录结构

安装锦帛浏览器后，会在安装目录下出现如下两个文件夹：



助手产品本地目录结构

EBankingAssistant文件夹中，为助手启动程序所在位置，在助手产品中，无法开启开发人员调试模式以获取内置浏览器详细信息。

Tapestry7文件夹中，即锦帛浏览器所在目录，启动后，可开启调试。

* 1. UA获取及对比

在启动的锦帛7浏览器及Chrome浏览器中，分别访问站点网站：<http://www.atool9.com/useragent.php即可回显详细UA>信息，也可打开百度首页f12抓取请求头进行查看，如下图所示：



锦帛浏览器UA信息



Chrome80 UA信息

* 1. Chrome41还是43？

通过上图可以发现，锦帛浏览器的内核版本，是基于Chromium41，似乎与之前所提基于43版本有出入：

通过与基础团队同事马秀红沟通，厂商开发**确实是基于43版本内核**，但是综合考虑Web前端代码中的41版本要求及其他因素，进行了处理**将版本显示调整为41**，服务器在获取UA时，显示为Chrome41版本。

经过沟通，我们自有提出的要求是，厂商的浏览器UA无论是内核版本，还是显示，**都应该是43，并加上自己的特殊标识**，配合交易银行WEB端完成UA识别，该问题后续需进一步确认。

1. Cookie和Session机制及处理差异

浏览器原理领域中，研究cookie及session机制至关重要。

* 1. Cookie及Session简要对比

What is cookie?

Cookies是作为HTTP传输的头信息的一部分发给客户机的“甜点”，为了解决HTTP协议无状态的缺陷。

What is session?

基于cookie，优化出的在客户端与服务器之间保持状态的解决方案。

具体来说cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。

对比Cookie及Session如下：

1、Cookie可以存储在浏览器或者本地，session只能存在服务器

2、Session比Cookie更具有安全性

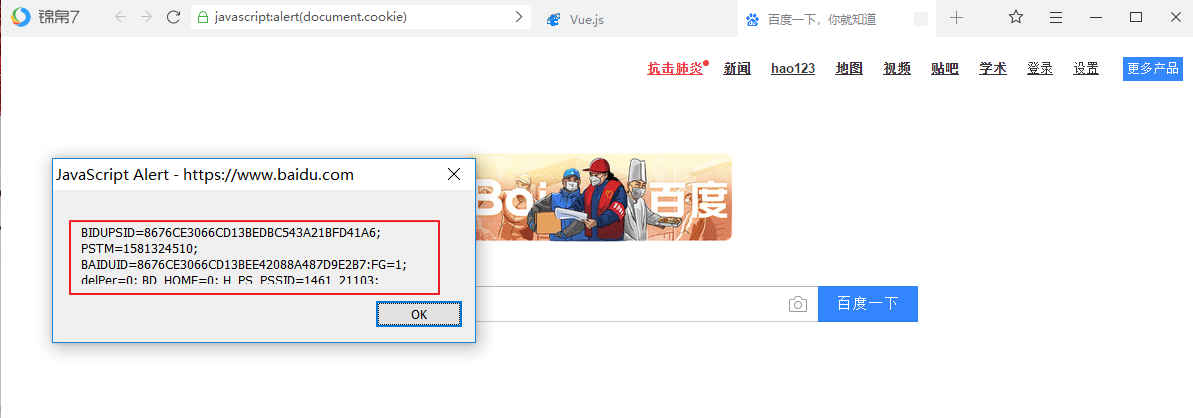
3、Session占用服务器性能，Session过多，增加服务器压力

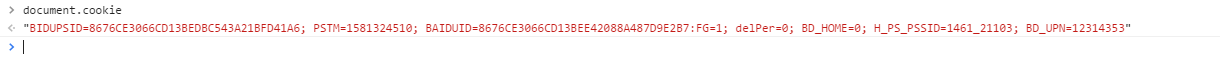
4、单个Cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个Cookie

* 1. 锦帛浏览器 VS Chrome 处理Cookie

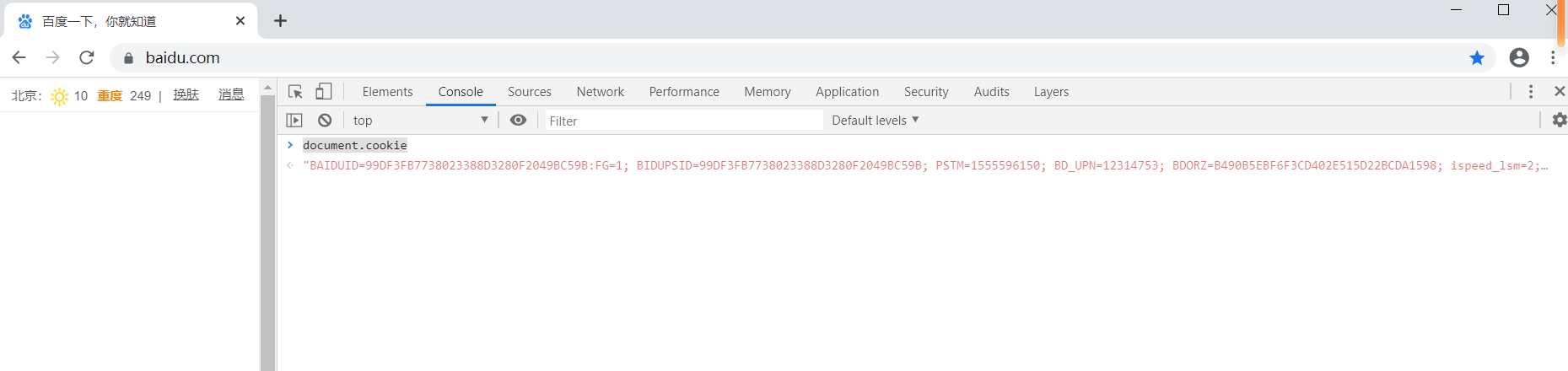
锦帛浏览器基于Chromium内核，其对Cookie及Session的处理机制同Chrome家族可以说大致相同，但是近日，谷歌发布Chrome80，其新特性包含了对Cookie的新机制，需重点关注。

分别查看锦帛与Chrome为百度首页保存的cookie详情：地址栏输入javascript:alert(document.cookie)或控制台输入document.cookie查看





锦帛浏览器百度首页Cookie



Chrome80 百度首页Cookie

* 1. Chrome80 Cookie新策略

在 Chrome 80 中，Chrome 会将没有声明 SameSite 值的 cookie 默认设置为SameSite=Lax。**只有采用SameSite=None; Secure设置的 cookie 可以从外部访问，前提是通过安全连接（即 HTTPS）访问。**

这个处理 cookie 文件方式的变化就是本次更新的第一项重大更改，它将改变以往用户隐私被滥用的现象，保证仅能通过 HTTPS 连接访问跨站 cookie。除了直接的安全优势外，明确声明跨站 cookie 还能提高透明度和用户选择。例如，浏览器可以为用户提供管理 cookie 时的精细控制，将仅可以在单个网站访问的 cookie 与可以在多个网站访问的 cookie 分开。

***后续考虑进行实践验证。***

1. Localstorage VS sessionstorage

Cookie即为HTML4提供的将用户存储在客户端上的方式，而由于cookie存储机制有很多缺点，HTML5不再使用它，转而使用改良后的Web Storage存储机制，即为localstorage和sessionstorage。本章节即研究cookie、localstorage和sessionstorage三者之间的详细区别，并探索验证锦帛浏览器对H5中新增的Web Storage两种存储机制支持情况，同Chrome对Web Storage支持情况做出对比。

* 1. cookie、localstorage与sessionstorage总结对比

cookie：

* 是网景公司的前雇员在1993年发明。它的主要用于保存登陆信息，比如登陆某个网站市场可以看到'记住密码’，这就是通过在cookie中存入一段辨别用户身份的数据来实现的。
* 一般不超过4K（因为每次http请求都会携带cookie、所以cookie只适合保存很小的数据，如会话标识）
* 一般由服务器生成，可以设置失效时间；若没有设置时间，关闭浏览器cookie失效，若设置了时间，cookie就会存放在硬盘里，过期才失效
* cookie在浏览器和服务器之间来回传递；如果使用cookie保存过多数据会造成性能问题
* 在所有同源窗口中都是共享的
* 场景：判断用户是否登录过网站，以便实现下次自动登录或记住密码；保存事件信息等

sessionStorage：

* 会话，是可以将一部分数据在当前会话中保存下来，刷新页面数据依旧存在。但是页面关闭后，sessionStorage中的数据就会被清空。
* 5M或者更大
* 仅在当前浏览器窗口关闭之前有效，关闭页面或者浏览器会被清除。将数据保存在session对象中。所谓session，是指用户在浏览某个网站时，从进入网站到浏览器关闭所经过的这段时间，也就是用户浏览这个网站所花费的时间。session对象可以用来保存在这段时间内所要求保存的任何数据。
* 仅在客户端（即浏览器）中保存，不参与和服务器的通信；不会自动把数据发送给服务器，仅在本地保存
* 在同一个浏览器窗口是共享的（不同浏览器、同一个页面也是不共享的）
* 场景：敏感账号一次性登录；单页面用的较多（sessionStorage 可以保证打开页面时 sessionStorage 的数据为空）

localStorage：

* 早在IE6时代就有一个userData的东西用于本地存储，而当时考虑到浏览器的兼容性，更通用的方案是使用flash。

如今localStorage被大多数浏览器所支持。

* 5M或者更大
* **永久有效**，窗口或者浏览器关闭也会一直保存，除非手动永久清除，因此用作持久数据。将数据保存在客户端本地的硬件设备(通常指硬盘，也可以是其他硬件设备)中，即使浏览器被关闭了，该数据仍然存在，下次打开浏览器访问网站时仍然可以继续使用。
* 仅在客户端（即浏览器）中保存，不参与和服务器的通信；不会自动把数据发送给服务器，仅在本地保存
* 在所有同源窗口中都是共享的
* 场景：常用于长期登录（判断用户是否已登录），适合长期保存在本地的数据
  1. 验证锦帛对Web Storage支持情况

编写如下js脚本并在锦帛浏览器的控制台执行，测试是否支持localstorage、sessionstorage及已使用存储空间大小：

function get\_cache\_size(t){

t = t == undefined ? "l" : t;

var obj = "";

if(t==='l'){

if(!window.localStorage) {

console.log('浏览器不支持localStorage');

}else{

obj = window.localStorage;

}

}else{

if(!window.sessionStorage) {

console.log('浏览器不支持sessionStorage');

}else{

obj = window.sessionStorage;

}

}

if(obj!==""){

var size = 0;

for(item in obj) {

if(obj.hasOwnProperty(item)) {

size += obj.getItem(item).length;

}

}

console.log('当前已用存储：' + (size / 1024).toFixed(2) + 'KB');

}

}

get\_cache\_size('l'); //localStorage当前大小

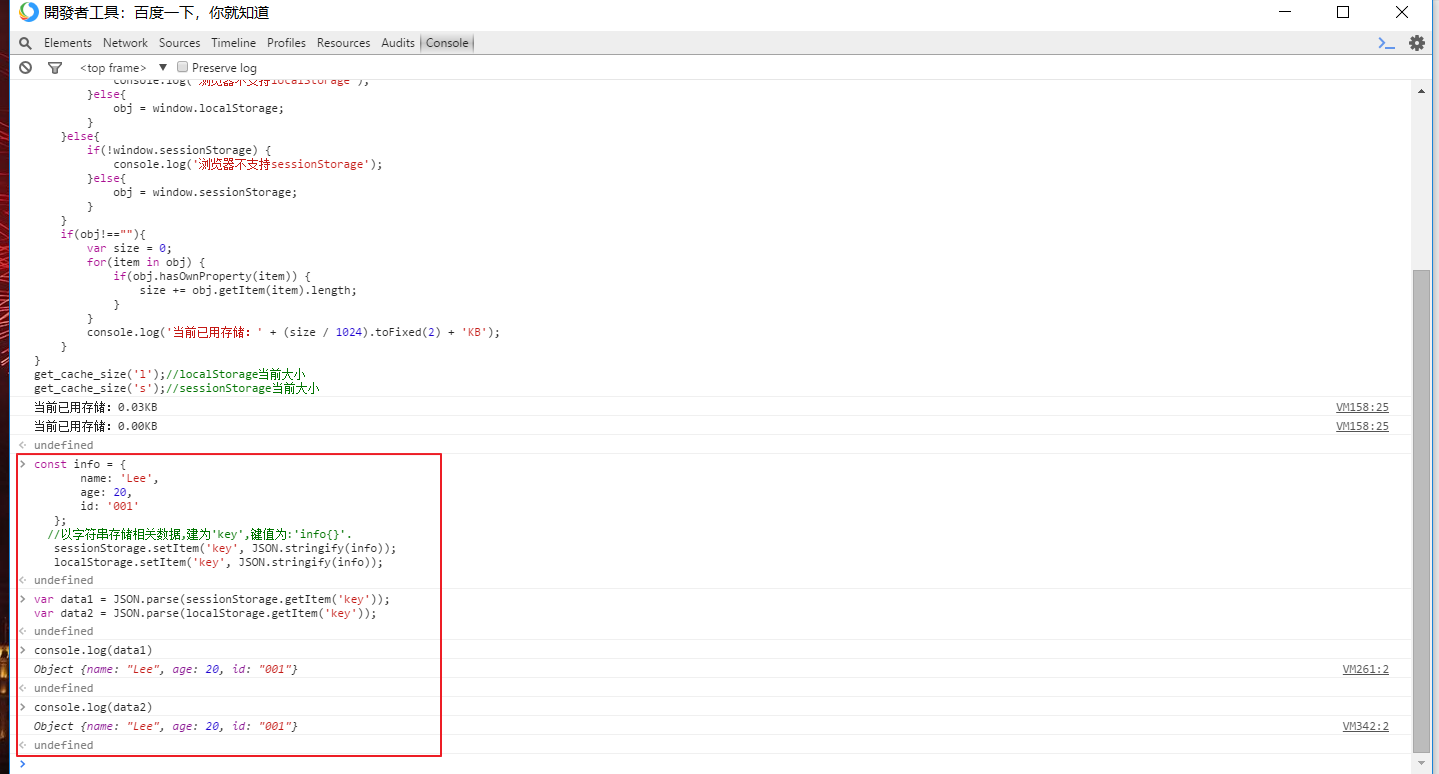
get\_cache\_size('s'); //sessionStorage当前大小



锦帛浏览器脚本执行结果

如图所示，锦帛浏览器支持localstorage及sessionstorage两种存储。

进一步验证两种存储的读写情况：



锦帛Web storage读写情况

上图所示，通过直接写入、读写，进一步验证锦帛浏览器支持localstorage及sessionstorage两种存储。

1. 浏览器渲染性能的benchmark测试

由于交易银行Web端开发主要使用Vuejs框架进行开发，所以我们主要测试对比锦帛浏览器及Chrome原生浏览器对Vuejs框架的渲染性能。

* 1. 浏览器渲染性能影响因素及优化

根据浏览器渲染原理，重排重绘位于整体渲染流水线前沿，当CSS属性使用不当时，会导致重排或重绘从而引起后续流水线的重新执行，影响客户体验

哪些CSS属性会引起重排或重绘：https://csstriggers.com/

开发过程中，可以采取如下优化策略来提升渲染性能：

（一）CSS属性读写分离：浏览器每次对元素样式进行读操作时，都必须进行一次重新渲染（重排 + 重绘），所以我们在使用JS对元素样式进行读写操作时，最好将两者分离开，先读后写，避免出现两者交叉使用的情况。最客观的解决方案，就是不用JS去操作元素样式。

（二）通过切换class或者style.csstext属性去批量操作元素样式

（三）DOM元素离线更新：当对DOM进行相关操作时，例、appendChild等都可以使用Document Fragment对象进行离线操作，带元素“组装”完成后再一次插入页面，或者使用display:none 对元素隐藏，在元素“消失”后进行相关操作。

（四）将没用的元素设为不可见：visibility: hidden，这样可以减小重绘的压力，必要的时候再将元素显示。

（五）压缩DOM的深度，一个渲染层内不要有过深的子元素，少用DOM完成页面样式，多使用伪元素或者box-shadow取代。

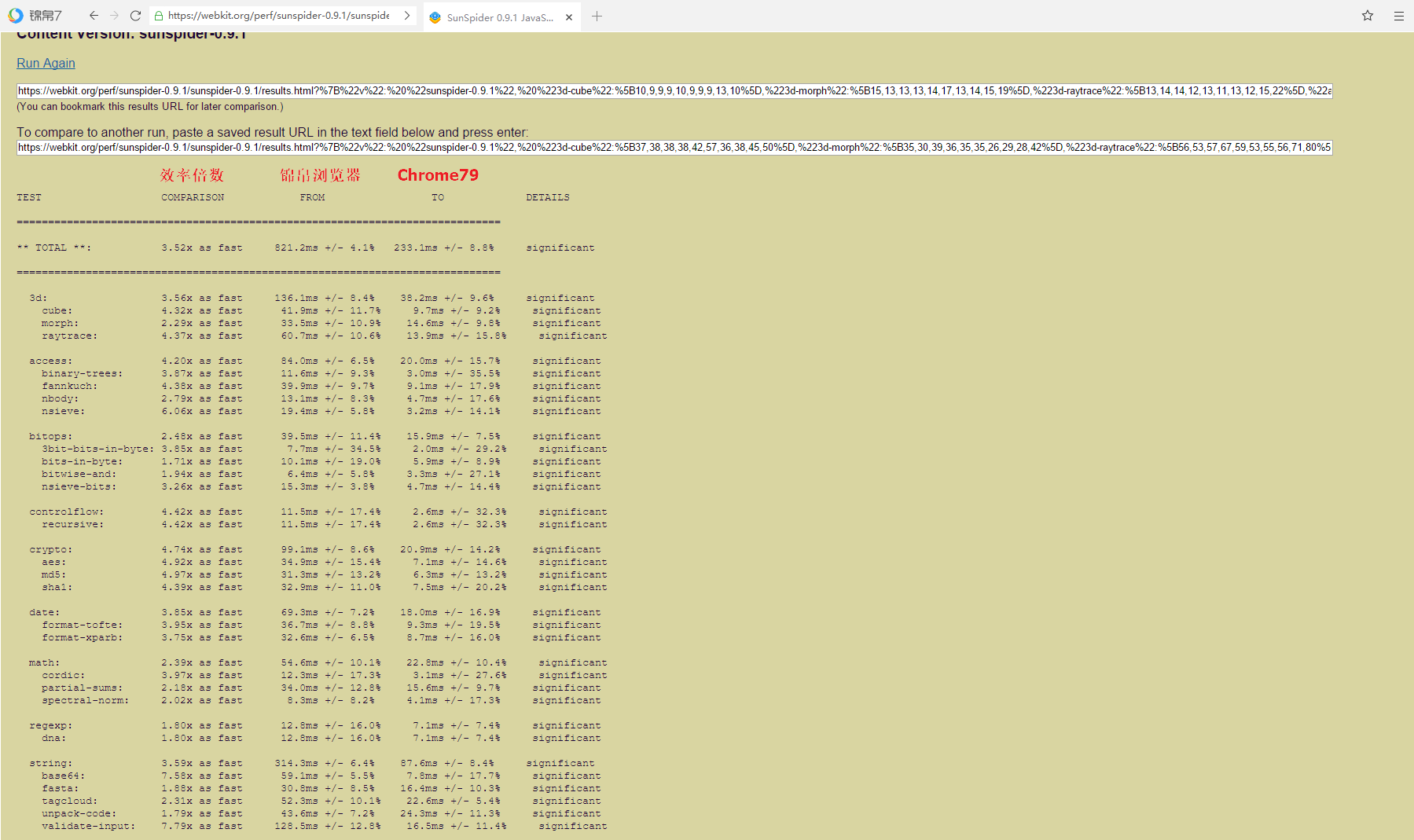
（六）图片在渲染前指定大小：因为img元素是内联元素，所以在加载图片后会改变宽高，严重的情况会导致整个页面重排，所以最好在渲染前就指定其大小，或者让其脱离文档流。

（七）对页面中可能发生大量重排重绘的元素单独触发渲染层，使用GPU分担CPU压力。（这项策略需要慎用，得着重考量以牺牲GPU占用率能否换来可期的性能优化。）

* 1. Javascript渲染性能对比

SunSpider 是 Mozilla 开发的 JavaScript 测试基准，是一款权威的专注于实际问题解决的测试软件。测试中，得分越低越好。

本小节主要使用SunSpider JavaScript Benchmark 测试对锦帛浏览器、Chrome79及最新版Chrome80的javascript渲染性能进行测试对比。



锦帛vs Chrome79

通过对比可以发现，Chrome79的JS运行效率，为锦帛浏览器的3.52倍。

将Chrome79升级至最新版Chrome80，重新构建对比：



锦帛vs Chrome80

通过对比可以发现，Chrome79的JS运行效率，为锦帛浏览器的4.00倍。

* 1. Vuejs渲染性能对比

js-framework-benchmark工具可以对目前很多流行框架的渲染性能进行测试，也是**VUE官方所推荐**的最佳测试工具，其代码开源托管在Github上：

<https://github.com/krausest/js-framework-benchmark>

我们使用js-framework-benchmark来测试锦帛浏览器与Chrome浏览器对Vuejs的渲染性能。

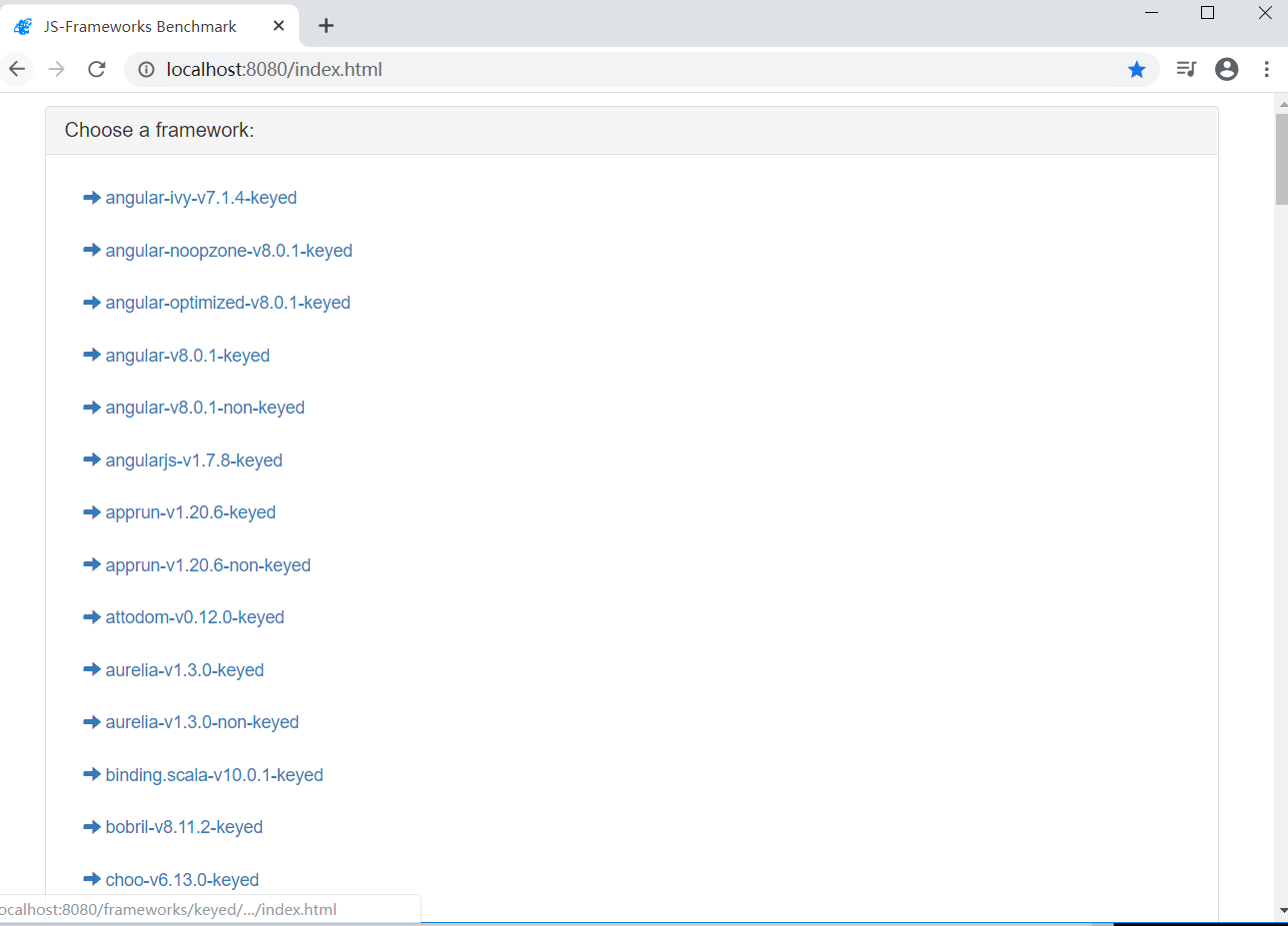
* + 1. 测试工具环境要求

框架要求环境配置：

1. Nodejs >= 10.0
2. Java >= 8
   * 1. 测试服务启动

启动服务：

1. js-framework-benchmark服务目录下，安装依赖npm install
2. npm start
3. 打开测试页 localhost:8080/index.html

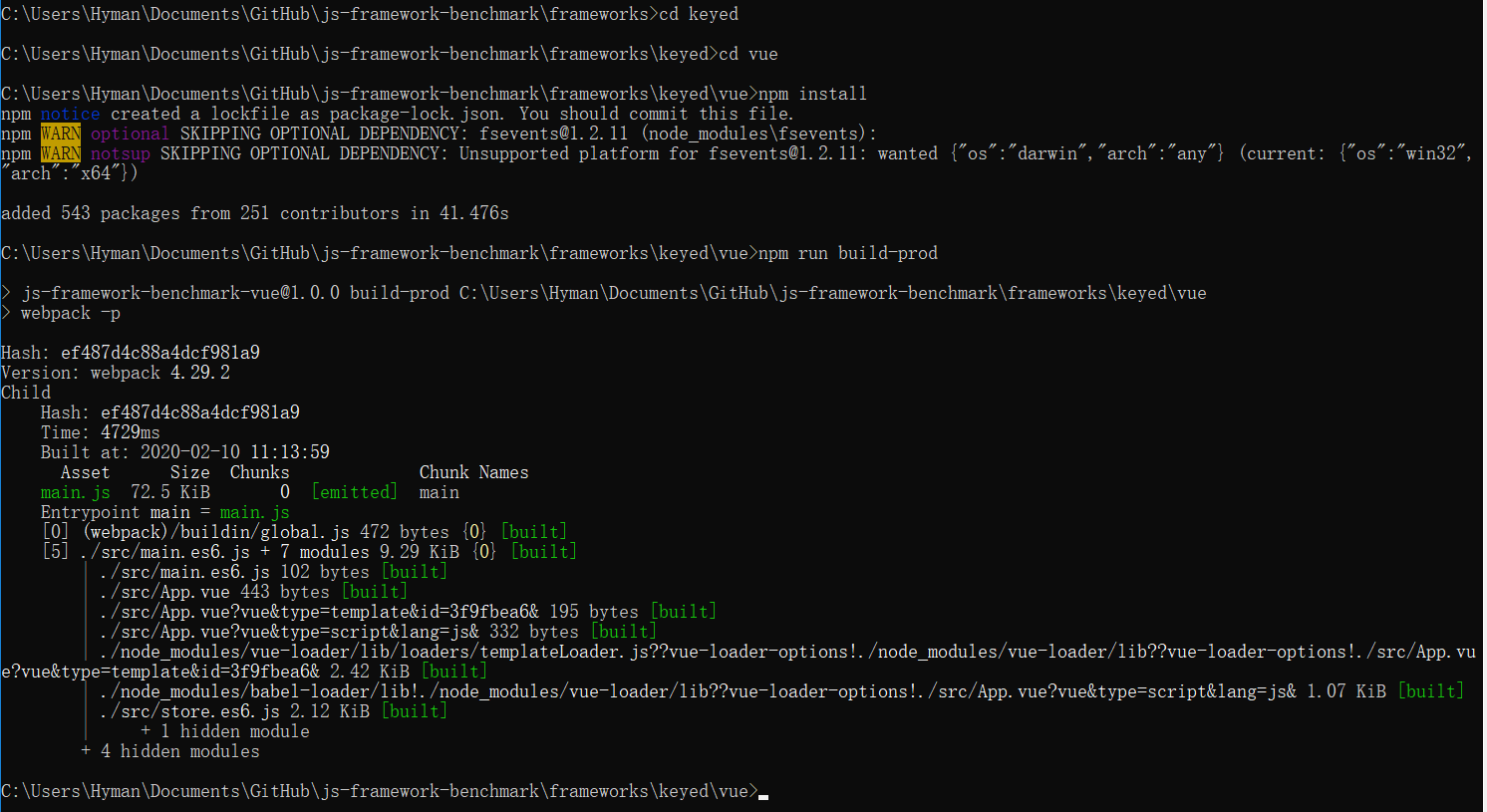


测试页

由上图可以看出测试框架支持众多主流框架，包括vue、angular等。注意，此时该服务不可停止，保持8080端口占用，为后续自动化测试必要条件。

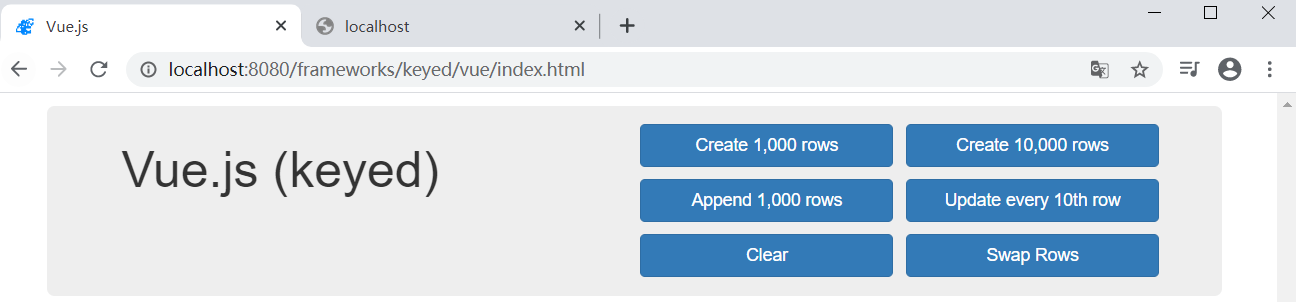
安装Vue框架依赖：

1. 进入目录 frameworks/keyed/vue
2. 安装依赖 npm install
3. 构建框架 npm run build-prod

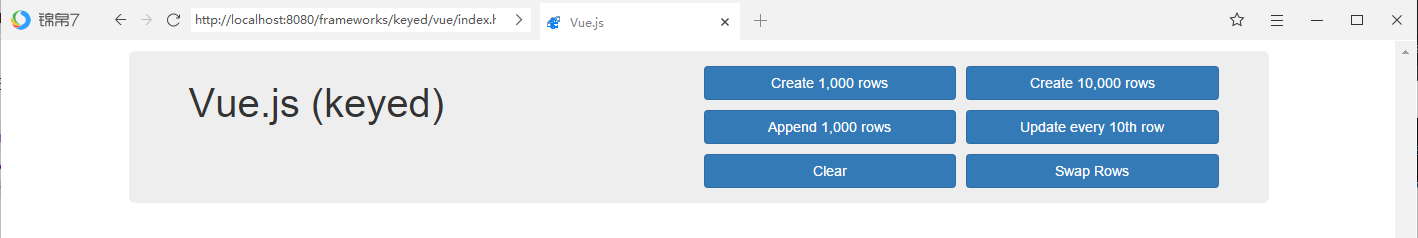


1. 在测试浏览器上，分别打开框架测试页查看是否显示正常:

<http://localhost:8080/frameworks/keyed/vue/index.html>



Chrome80 Vuejs测试页



锦帛浏览器 Vuejs测试页

从测试页可以看出测试项包括：创建1000行，创建10000行，增添1000行，每隔10行进行更新等测试项。

使用测试框架对Vuejs执行效率进行测试：

1. 回到js-framework-benchmark服务目录
2. 进入webdriver-ts路径下
3. 安装依赖 npm install
4. 编译benchmark驱动 npm run build-prod
5. 运行测试框架 npm run bench keyed/vue

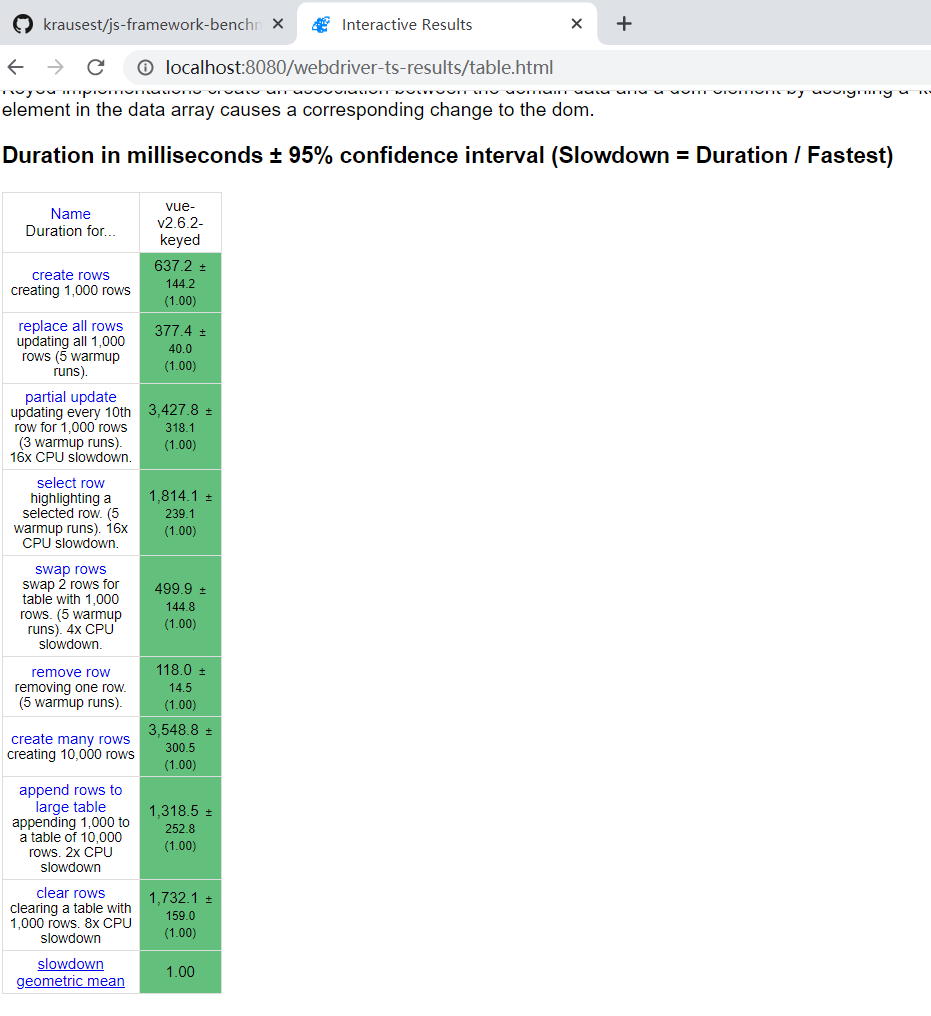
注意事项：上步启动的8080服务端口不可停止，此处测试框架调用必需。

1. 静待运行完成

测试结果导出

1. 回到js-framework-benchmark服务目录
2. 进入webdriver-ts-results路径下
3. 安装依赖 npm install
4. 回到webdriver-ts路径下
5. 运行报告查看服务 npm run results

报表如下所示：<http://localhost:8080/webdriver-ts-results/table.html>



Chrome vuejs渲染性能

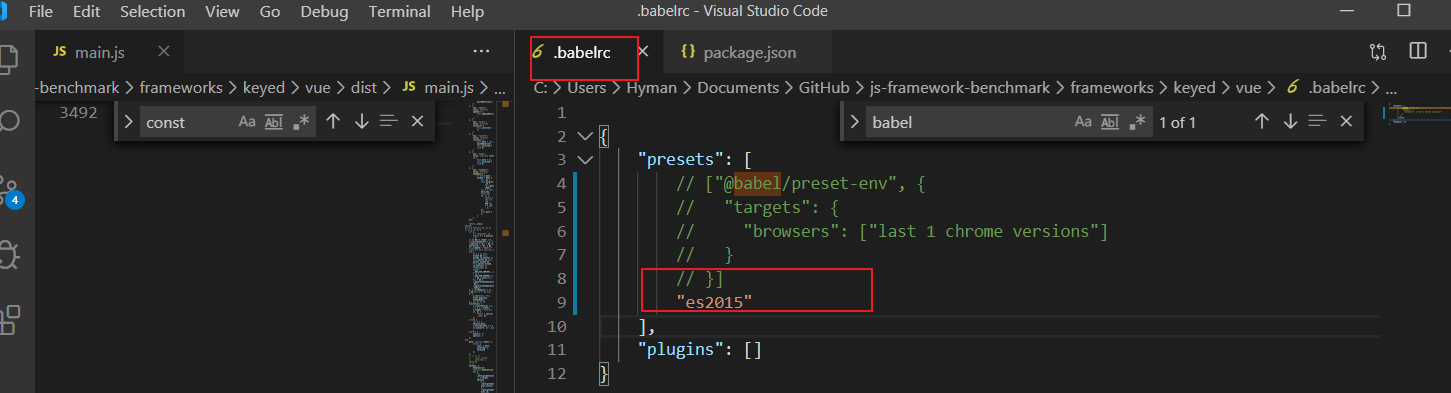
* + 1. 遇到的问题及解决方法

4.3.3.1 框架ES6语法兼容问题

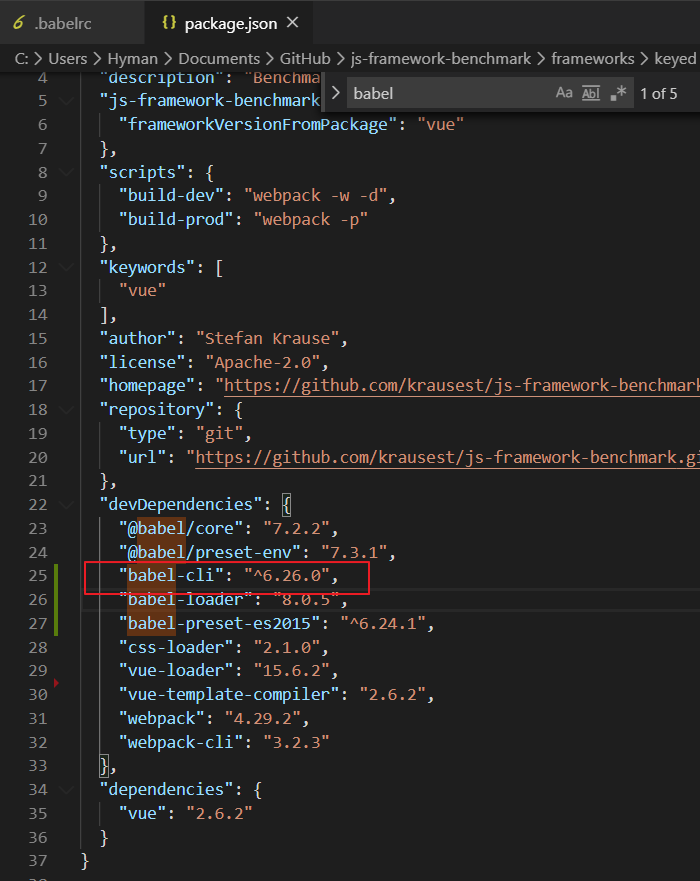
由于js-framework-benchmark框架测试JS文件基于ES6语法，故在41内核的锦帛上报出兼容性问题，需采用手动改写方式或babel工具进行转换。

使用Babel进行语法转换步骤：

1. 回到js-framework-benchmark服务目录
2. 全局安装babel工具npm install -g babel-cli
3. 继续安装npm install --save-dev babel-preset-es2015 babel-cli
4. 制定语法转换目标版本：新建.babelrc，在根目录下新建.babelrc文件，并打开录入下面的代码



Babel配置

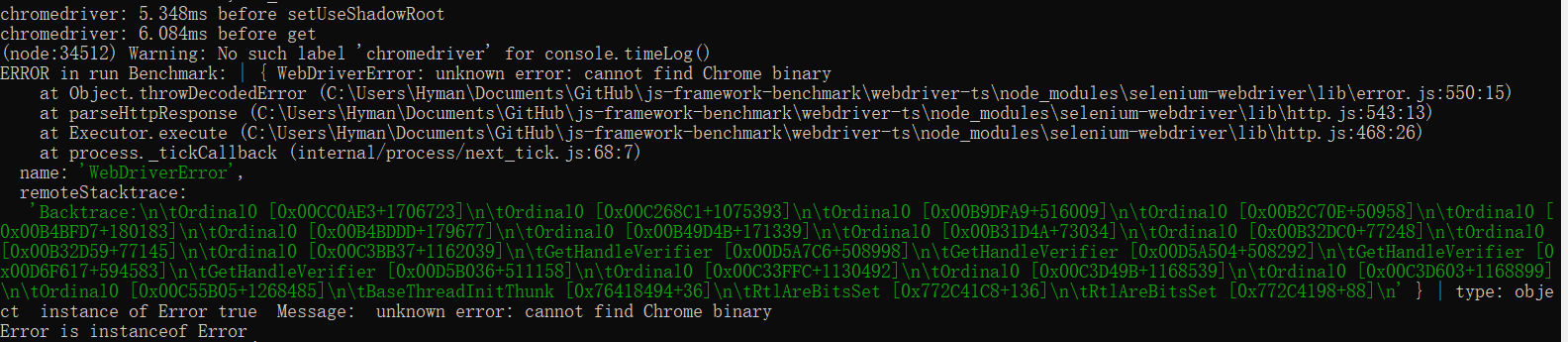


Babel配置

重新打包编译运行即可。

* + - 1. 锦帛浏览器无法自动化测试问题

该测试工具，依赖于selenium，可实现在Chrome内核浏览器上的脚本自动化测试等功能，然而在调起锦帛浏览器时，报出下方错误：



锦帛浏览器测试报错

根据查询资料及咨询专家庞健同事，该测试工具在Chromium内核上均应支持，该问题正通过厂商管控同事马秀红确认锦帛是否支持selenium。

1. 浏览器漏洞分析
   1. 浏览器漏洞类型汇总

* SQL注入漏洞

SQL注入攻击（SQL Injection），简称注入攻击、SQL注入，被广泛用于非法获取网站控制权，是发生在应用程序的数据库层上的安全漏洞。在设计程序，忽略了对输入字符串中夹带的SQL指令的检查，被数据库误认为是正常的SQL指令而运行，从而使数据库受到攻击，可能导致数据被窃取、更改、删除，以及进一步导致网站被嵌入恶意代码、被植入后门程序等危害。

通常情况下，SQL注入的位置包括：

（1）表单提交，主要是POST请求，也包括GET请求；

（2）URL参数提交，主要为GET请求参数；

（3）Cookie参数提交；

（4）HTTP请求头部的一些可修改的值，比如Referer、User\_Agent等；

（5）一些边缘的输入点，比如.mp3文件的一些文件信息等。

SQL注入的危害不仅体现在数据库层面上，还有可能危及承载数据库的操作系统；如果SQL注入被用来挂马，还可能用来传播恶意软件等

* 跨站脚本漏洞

跨站脚本攻击（Cross-site scripting，通常简称为XSS）发生在客户端，可被用于进行窃取隐私、钓鱼欺骗、窃取密码、传播恶意代码等攻击。

XSS攻击使用到的技术主要为HTML和Javascript，也包括VBScript和ActionScript等。XSS攻击对WEB服务器虽无直接危害，但是它借助网站进行传播，使网站的使用用户受到攻击，导致网站用户帐号被窃取，从而对网站也产生了较严重的危害。

XSS类型包括：

（1）非持久型跨站：即反射型跨站脚本漏洞，是目前最普遍的跨站类型。跨站代码一般存在于链接中，请求这样的链接时，跨站代码经过服务端反射回来，这类跨站的代码不存储到服务端（比如数据库中）。上面章节所举的例子就是这类情况。

（2）持久型跨站：这是危害最直接的跨站类型，跨站代码存储于服务端（比如数据库中）。常见情况是某用户在论坛发贴，如果论坛没有过滤用户输入的Javascript代码数据，就会导致其他浏览此贴的用户的浏览器会执行发贴人所嵌入的Javascript代码。

（3）DOM跨站（DOM XSS）：是一种发生在客户端DOM（Document Object Model文档对象模型）中的跨站漏洞，很大原因是因为客户端脚本处理逻辑导致的安全问题。

* 弱口令漏洞

弱口令(weak password) 没有严格和准确的定义，通常认为容易被别人（他们有可能对你很了解）猜测到或被破解工具破解的口令均为弱口令。

* HTTP报头追踪漏洞

HTTP/1.1（RFC2616）规范定义了HTTP TRACE方法，主要是用于客户端通过向Web服务器提交TRACE请求来进行测试或获得诊断信息。当Web服务器启用TRACE时，提交的请求头会在服务器响应的内容（Body）中完整的返回，其中HTTP头很可能包括Session Token、Cookies或其它认证信息。攻击者可以利用此漏洞来欺骗合法用户并得到他们的私人信息。该漏洞往往与其它方式配合来进行有效攻击，由于HTTP TRACE请求可以通过客户浏览器脚本发起（如XMLHttpRequest），并可以通过DOM接口来访问，因此很容易被攻击者利用。

* Struts2远程命令执行漏洞

Apache Struts是一款建立Java web应用程序的开放源代码架构。Apache Struts存在一个输入过滤错误，如果遇到转换错误可被利用注入和执行任意Java代码。

网站存在远程代码执行漏洞的大部分原因是由于网站采用了Apache Struts Xwork作为网站应用框架，由于该软件存在远程代码执高危漏洞，导致网站面临安全风险。CNVD处置过诸多此类漏洞，例如：“GPS车载卫星定位系统”网站存在远程命令执行漏洞(CNVD-2012-13934)；Aspcms留言本远程代码执行漏洞（CNVD-2012-11590）等。

* 框架钓鱼漏洞（框架注入漏洞）

框架注入攻击是针对Internet Explorer 5、Internet Explorer 6、与 Internet Explorer 7攻击的一种。这种攻击导致Internet Explorer不检查结果框架的目的网站，因而允许任意代码像Javascript或者VBScript跨框架存取。这种攻击也发生在代码透过多框架注入，肇因于脚本并不确认来自多框架的输入。这种其他形式的框架注入会影响所有的不确认不受信任输入的各厂商浏览器和脚本。

* 文件上传漏洞

文件上传漏洞通常由于网页代码中的文件上传路径变量过滤不严造成的，如果文件上传功能实现代码没有严格限制用户上传的文件后缀以及文件类型，攻击者可通过 Web 访问的目录上传任意文件，包括网站后门文件（webshell），进而远程控制网站服务器。

因此，在开发网站及应用程序过程中，需严格限制和校验上传的文件，禁止上传恶意代码的文件。同时限制相关目录的执行权限，防范webshell攻击。

* 应用程序测试脚本泄露

由于测试脚本对提交的参数数据缺少充分过滤，远程攻击者可以利用洞以WEB进程权限在系统上查看任意文件内容。防御此类漏洞通常需严格过滤提交的数据，有效检测攻击。

* 私有IP地址泄露漏洞

IP地址是网络用户的重要标示，是攻击者进行攻击前需要了解的。获取的方法较多，攻击者也会因不同的网络情况采取不同的方法，如：在局域网内使用Ping指令，Ping对方在网络中的名称而获得IP；在Internet上使用IP版的QQ直接显示。最有效的办法是截获并分析对方的网络数据包。攻击者可以找到并直接通过软件解析截获后的数据包的IP包头信息，再根据这些信息了解具体的IP。

针对最有效的“数据包分析方法”而言，就可以安装能够自动去掉发送数据包包头IP信息的一些软件。不过使用这些软件有些缺点，譬如：耗费资源严重，降低计算机性能；访问一些论坛或者网站时会受影响；不适合网吧用户使用等等。现在的个人用户采用最普及隐藏IP的方法应该是使用代理，由于使用代理服务器后，“转址服务”会对发送出去的数据包有所修改，致使“数据包分析”的方法失效。一些容易泄漏用户IP的网络软件(QQ、MSN、IE等)都支持使用代理方式连接Internet，特别是QQ使用“ezProxy”等代理软件连接后，IP版的QQ都无法显示该IP地址。虽然代理可以有效地隐藏用户IP，但攻击者亦可以绕过代理，查找到对方的真实IP地址，用户在何种情况下使用何种方法隐藏IP，也要因情况而论。

* 未加密登录请求

由于Web配置不安全，登陆请求把诸如用户名和密码等敏感字段未加密进行传输，攻击者可以窃听网络以劫获这些敏感信息。建议进行例如SSH等的加密后再传输。

* 敏感信息泄露漏洞

SQL注入、XSS、目录遍历、弱口令等均可导致敏感信息泄露，攻击者可以通过漏洞获得敏感信息。针对不同成因，防御方式不同。

* 1. Chrome漏洞汇总

汇总如下漏洞，采取了国内外网站联合查询的方式，参考链条及方法如下：

* [**https://www.cnvd.org.cn/flaw/list.htm**](https://www.cnvd.org.cn/flaw/list.htm)

可以在这个页面查询已公布修复的chrome漏洞

下面的参考网址**需要翻墙**:

* [**https://www.cvedetails.com/**](https://www.cvedetails.com/)

这个网站可以根据上个链接查询到的chrome的bug中的CVE代码查询

* https://chromereleases.googleblog.com/2019/03/stable-channel-update-for-desktop\_12.html

可以查看chrome更新的blog说明

* https://crbug.com/914983 可以跳转到chromium的bug的详情页面，会相对应的描述
* <https://cs.chromium.org/chromium/> 查看源码
* <http://lists.opensuse.org/opensuse-security-announce/2019-06/msg00085.html>

收集到的Chrome漏洞如下：

1. 整数溢出漏洞（CNVD-2019-23135）CVE-2019-5792

影响产品：Google, Inc. Chrome <73.0.3683.75

漏洞描述：Google Chrome 73.0.3683.75之前版本中的PDFium存在整数溢出漏洞。攻击者可利用该漏洞通过精心制作的PDF文件可能执行越界内存访问。

验证是否修复手段:

2. 类型混淆漏洞（CNVD-2020-00274）CVE-2019-13730

影响产品：Google Chrome <79.0.3945.79

漏洞描述：Google Chrome存在类型混淆漏洞。攻击者可借助特制HTML页面利用该漏洞造成堆损坏。

验证是否修复手段:

3. 越界读取漏洞（CNVD-2019-43386） CVE-2019-5849

影响产品：Google Chrome <75.0.3770.80

漏洞描述：Google Chrome 75.0.3770.80之前版本中的Skia存在越界读取漏洞。远程攻击者可利用该漏洞通过特制HTML页面从进程内存中获取敏感信息。

验证是否修复手段:

4. 代码注入漏洞（CNVD-2019-38244） CVE-2019-13714

影响产品：Google Chrome <78.0.3904.70

漏洞描述：Google Chrome 78.0.3904.70之前版本中存在代码注入漏洞，该漏洞源于在处理CSS文件时，程序没有进行正确的输入验证。攻击者可利用借助特制的Web页面利用该漏洞执行任意JavaScript代码。

Chromium issues地址: https://bugs.chromium.org/p/chromium/issues/detail?id=982812&colspec=ID%20Status%20Owner%20Summary%20OS%20Modified&q=CVE-2019-13714&can=7

VERSION

Chrome Version: 75 stable

Operating System: Windows 10

生成原因:

Color Enhancer extension has following code in content script.

var baseUrl = window.location.href.replace(window.location.hash, '');

style = document.createElement('style');

style.id = STYLE\_ID;

style.setAttribute('type', 'text/css');

style.innerHTML = cssTemplate.replace(/#/g, baseUrl + '#'); // <-- Taking unverified URL into style

document.head.appendChild(style);

实现步骤：

1. Install https://chrome.google.com/webstore/detail/color-enhancer/ipkjmjaledkapilfdigkgfmpekpfnkih
2. Go to https://www.google.com/;background-image:url('https://attack.shhnjk.com/test.jpeg')
3. Observe that CSS is injected into Google.

验证是否修复手段:

5. 地址栏欺骗漏洞（CNVD-2016-06122） CVE-2016-5141

影响产品：Google Chrome <52.0.2743.116

漏洞描述：Google Chrome存在地址栏欺骗漏洞。允许攻击者通过涉及一个初始空文档的临时网址的向量，欺骗地址栏。

漏洞复现技术:

验证手段:

6. 脚本注入漏洞（CNVD-2016-07213）CVE-2016-5165

影响产品：Google Chrome <53.0.2785.89

漏洞描述：Google Chrome 53.0.2785.89之前的版本中存在脚本注入漏洞。攻击者可利用该漏洞在浏览器上下文中执行任意代码，绕过安全限制，执行未授权操作，或造成拒绝服务。

验证是否修复手段:

7. 越边界访问漏洞（CNVD-2018-10575）EDB-ID:42286

影响产品：Google Chrome <65.0.3325.146

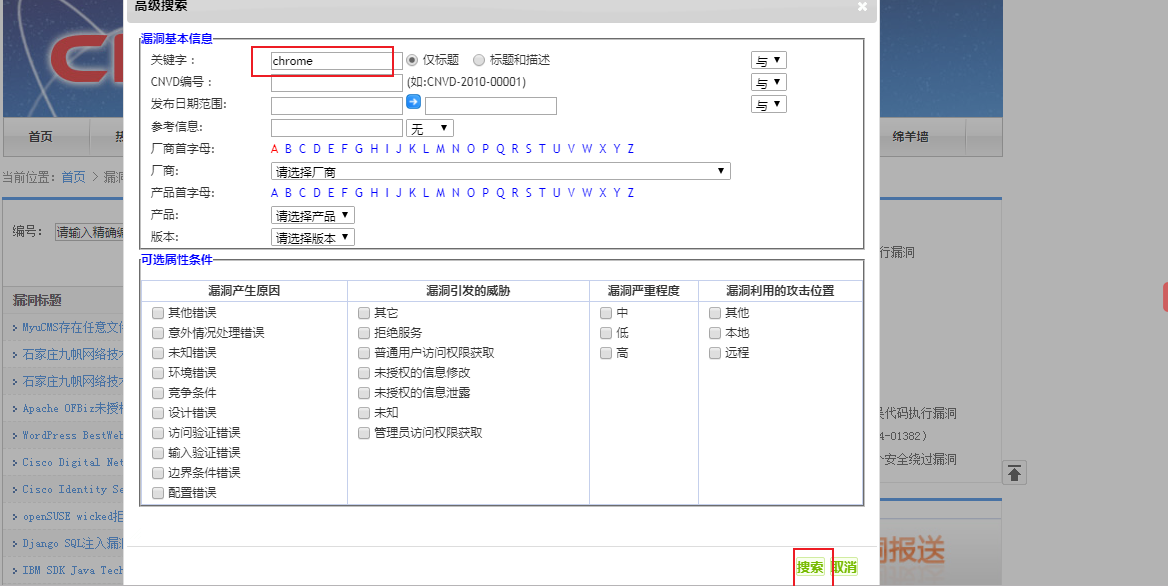
漏洞描述：Google Chrome存在越边界访问漏洞。攻击者可利用此漏洞造成越边界读取和写入，导致内联字段（如lastIndex）更改为字典属性，导致进一步攻击。

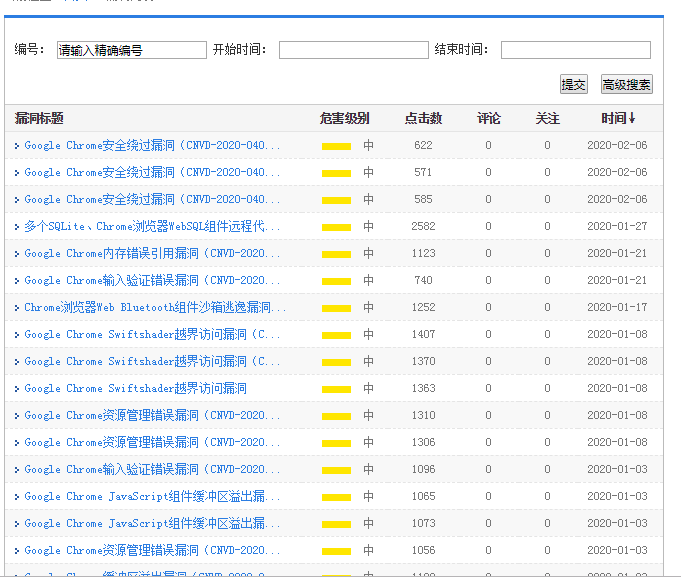
验证是否修复手段

针对各个漏洞，目前已查明一些轻量级可复现漏洞，实现尚需时间，验证可行性正在研究。

* 1. chrome漏洞查找及复现步骤

1. 已修复漏洞
   1. 进入**<https://www.cnvd.org.cn/flaw/list.htm>**网站
   2. 进入此页面之后，点击高级搜索，输入关键字chrome点击

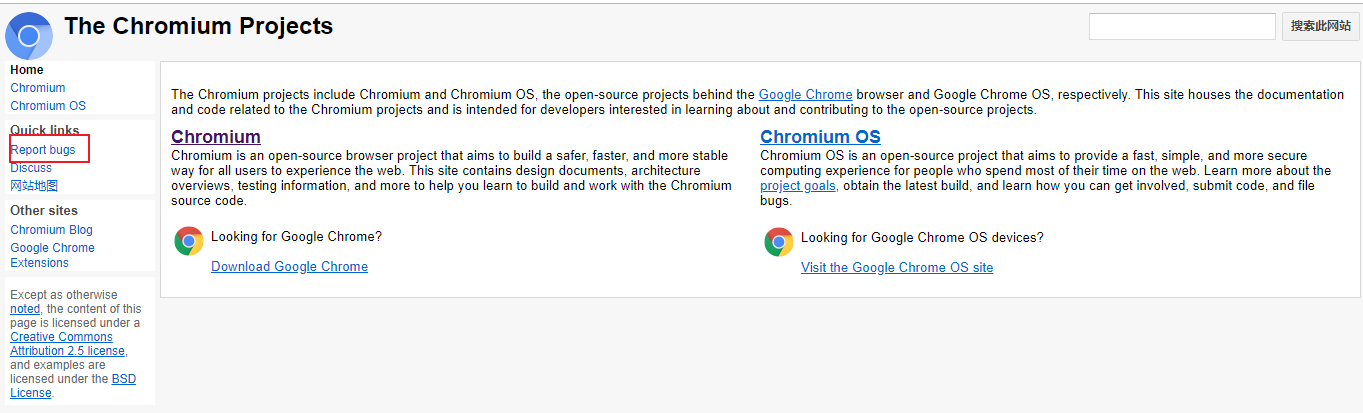




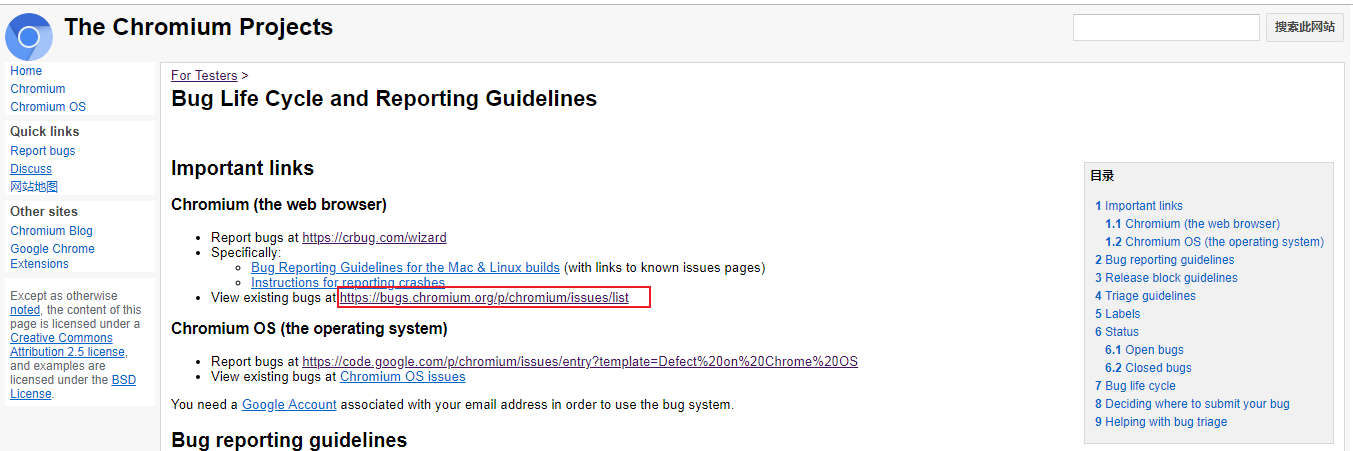
1.3点击其中的一条可以直接进入详情，其中会标明影响的产品和CVE id，漏洞类型，漏洞描述等信息。



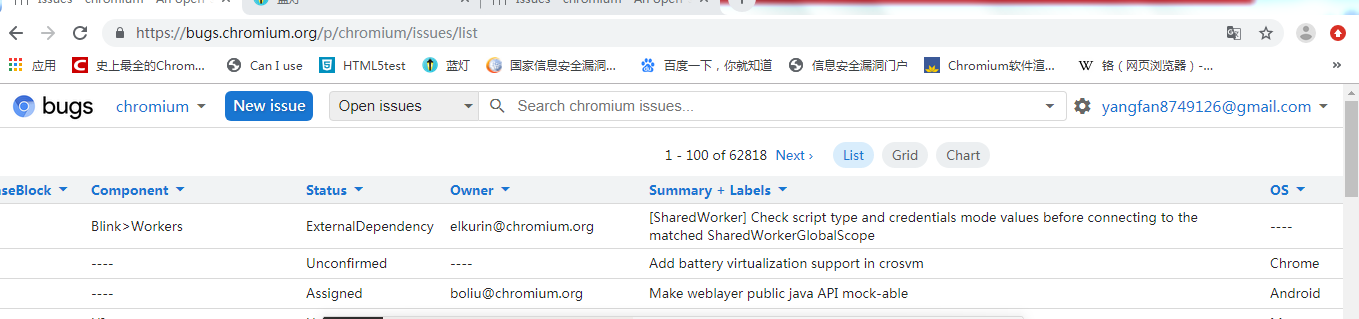
1.4 使用翻墙软件,打开官网<https://www.chromium.org/>



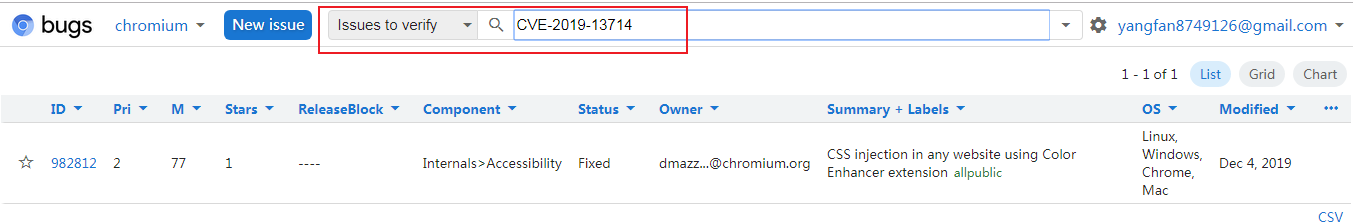
* 1. 点击[Report bugs](https://www.chromium.org/for-testers/bug-reporting-guidelines)报告错误进入下图



1.6 点击标红的连接，进入到下面的页（<https://bugs.chromium.org/p/chromium/issues/list>）



选择issues to verify 并在输入框中输入CVE编号，就会查询相对应的条目，点击列表中的条目,进入到详情页面



进入到详情页面



**翻译如下:**

**漏洞细节:**

颜色增强器扩展在内容脚本中有以下代码。

var baseUrl = window.location.href.replace(window.location)。哈希,”);

style= document.createElement(“style”);

style.id = STYLE\_ID;

style.setAttribute(“类型”,“文字/ css”);

style.innerHTML = cssTemplate。替换(/#/g, baseUrl + '#');// <——将未验证的URL放入样式中

document.head.appendChild(style);

**版本：**

Chrome版本:75稳定

操作系统:Windows 10

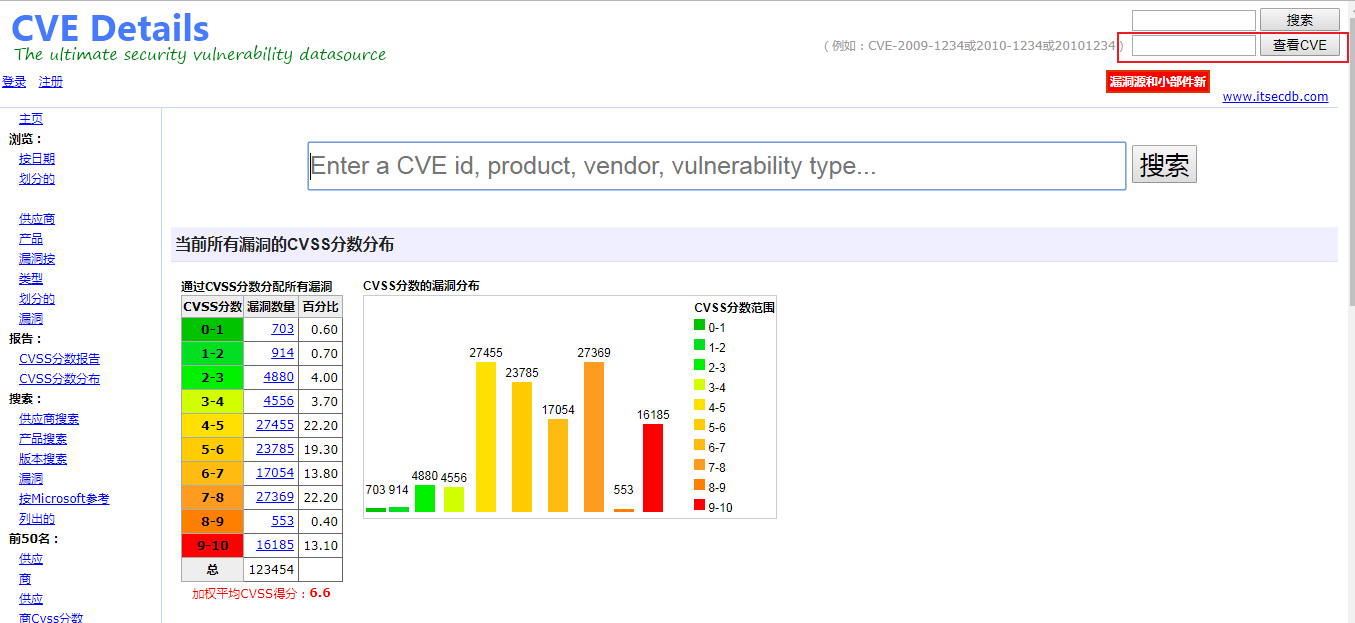
**实现情况:**

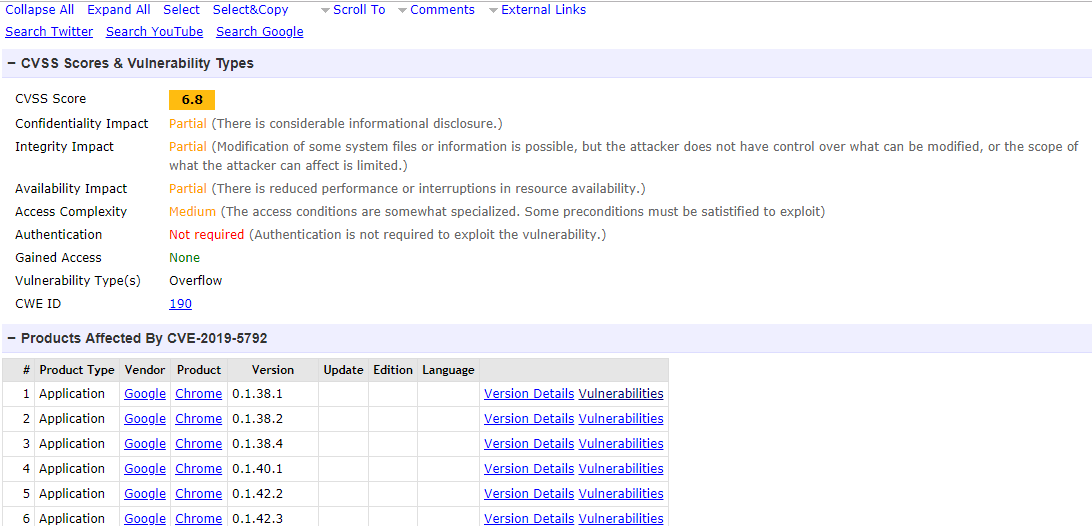
1. 安装https://chrome.google.com/webstore/detail/color-enhancer/ipkjmjaledkapilfdigkgfmpekpfnkih

2. 去https://www.google.com/ ');背景图片:url (https://attack.shhnjk.com/test.jpeg

3. 注意，CSS被注入到谷歌中。

若要查询此漏洞更多的信息可以翻墙进入到<https://www.cvedetails.com/>网站，此网站CVE不一定都能查询到



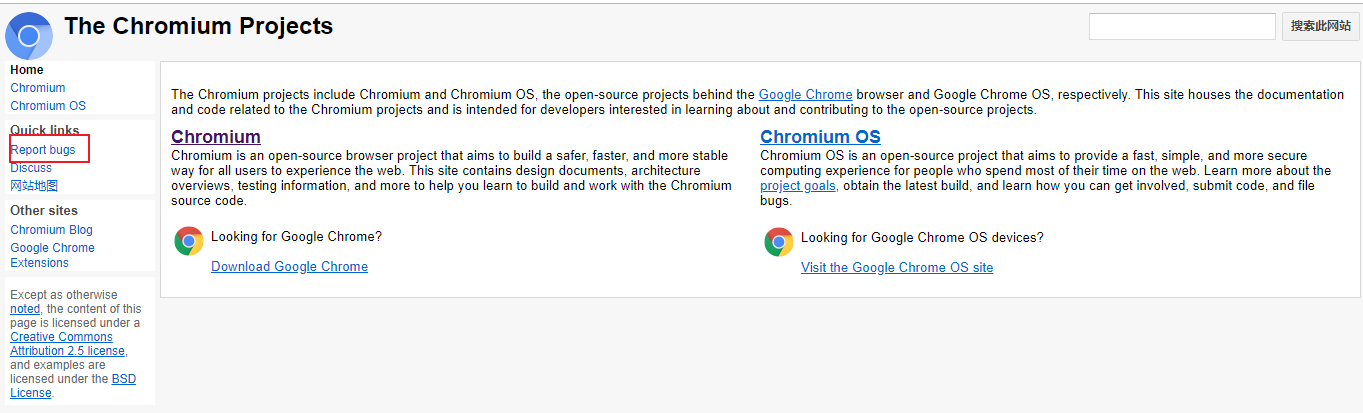
在上图标红的地方可以输入1.4步骤中的CVE id查询相关信息，可以查看



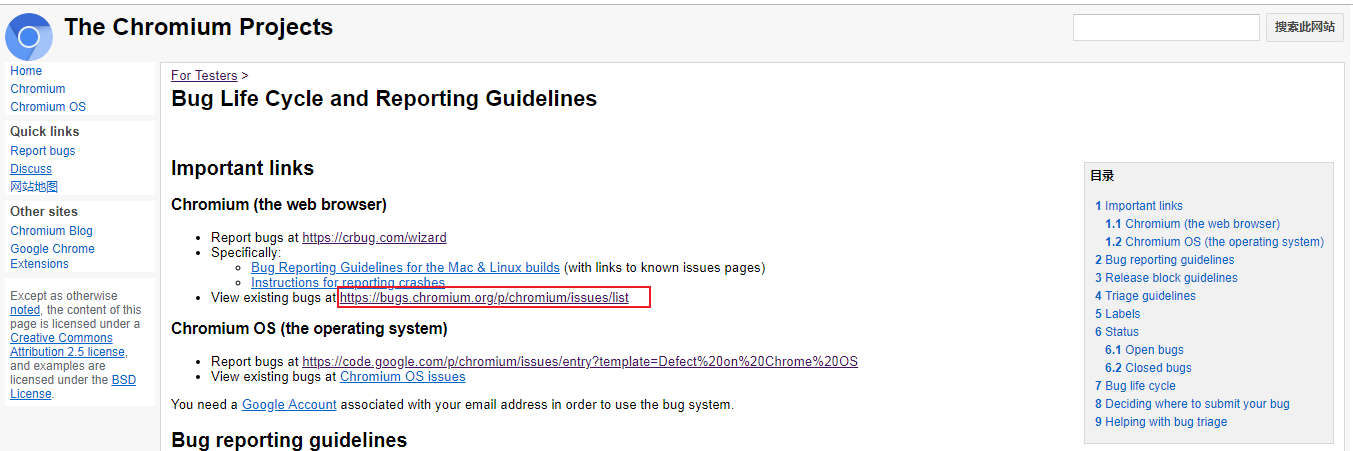
根据上图示意可以查看相对应的网址

2. Chromium官网提出的open issues

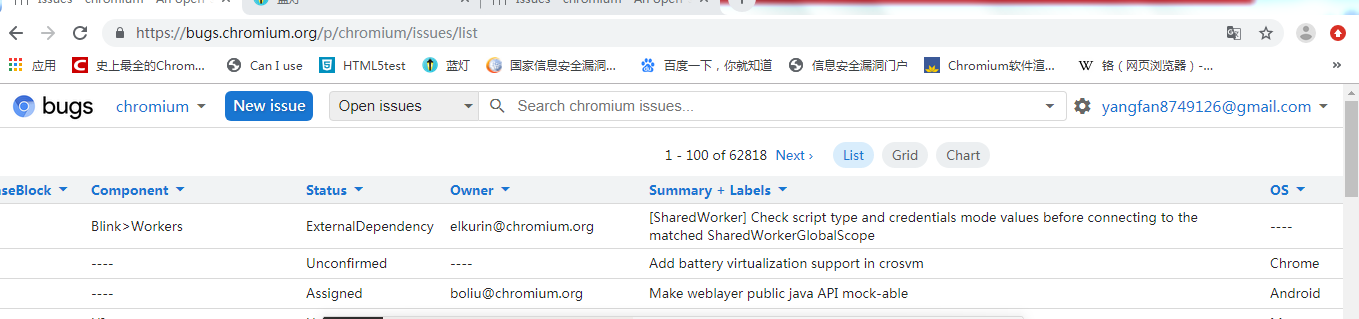
2.1 使用翻墙软件,打开官网<https://www.chromium.org/>



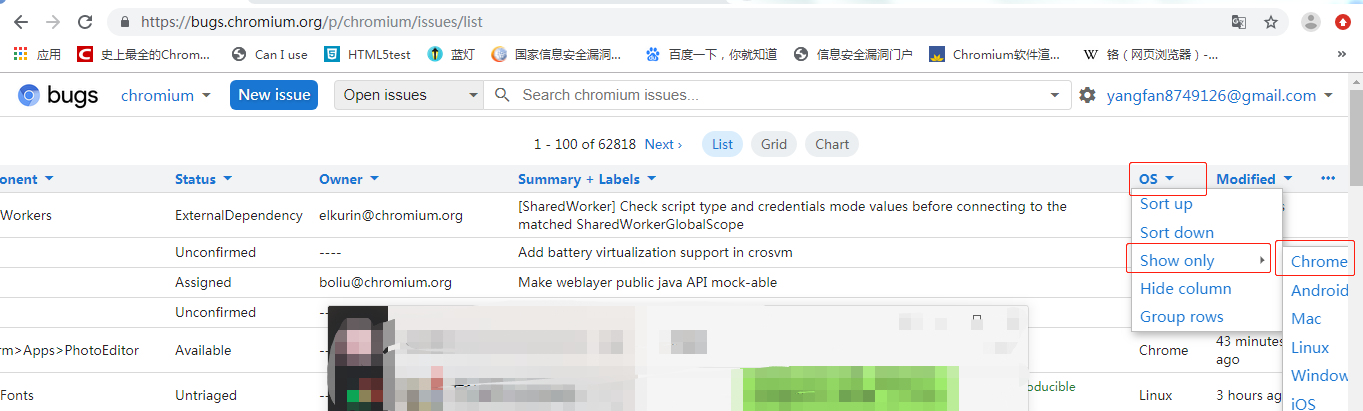
2.2点击[Report bugs](https://www.chromium.org/for-testers/bug-reporting-guidelines)报告错误进入下图



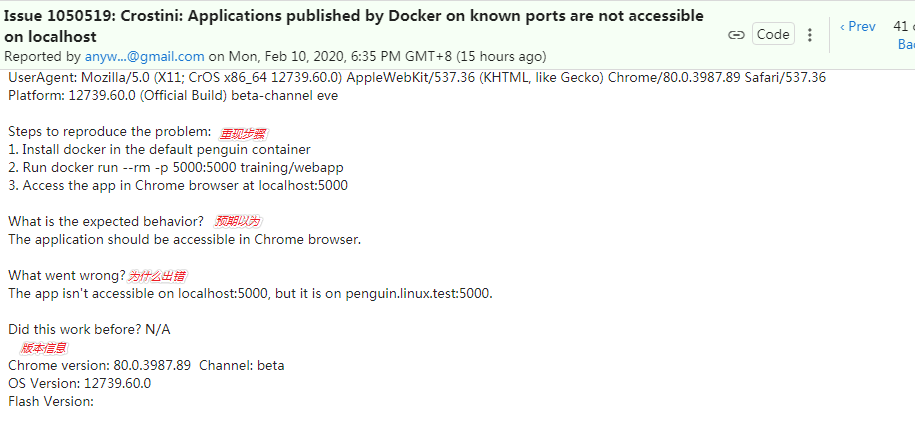
2.3点击标红的连接，进入到下面的页面（<https://bugs.chromium.org/p/chromium/issues/list>）



2.4如下入所以可以列出关于chrome的



2.5点击一条进入详情,可以进入到下面的页面



**翻译如下:**

**重现问题的步骤:**

1. 在默认的penguin容器中安装docker

2. 运行docker Run——rm -p 5000:5000训练/webapp

3.在本地主机:5000访问Chrome浏览器中的应用程序

**预期的行为是什么?**

这个应用程序应该可以在Chrome浏览器中访问。

**到底是哪里出了错?**

该应用程序无法在localhost:5000上访问，但它在企鹅。linux.test:5000上。

以前有用吗?N /一个

Chrome版本:80.0.3987.89频道:测试版

操作系统版本:12739.60.0

Flash版本: