内置浏览器路径：

C:\Program Files (x86)\BOCBank\Tapestry7

访问网页：

http://www.atool9.com/useragent.php



对cookie和session的处理差异

Cookie Localstorage和sessionstorage:

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

HTML5的WebStorage提供了两种API：localStorage（本地存储）和sessionStorage（会话存储）

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）

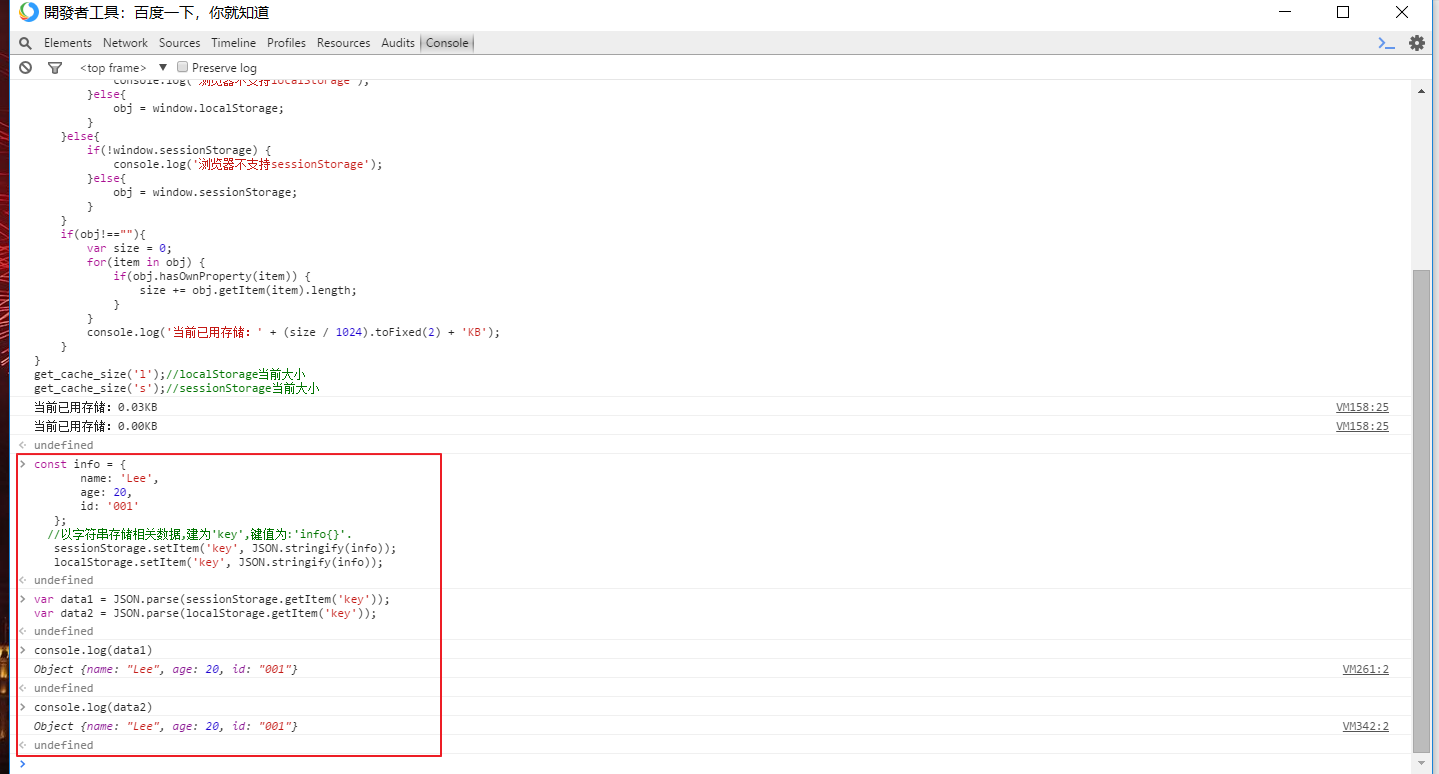
localStorage：

* 是HTML5标准中新加入的技术，当然早在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

探索锦帛浏览器是否支持localstorage及sessionstorage：



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



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。

浏览器渲染性能

优化浏览器渲染性能：减少重排

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

优化策略

（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。

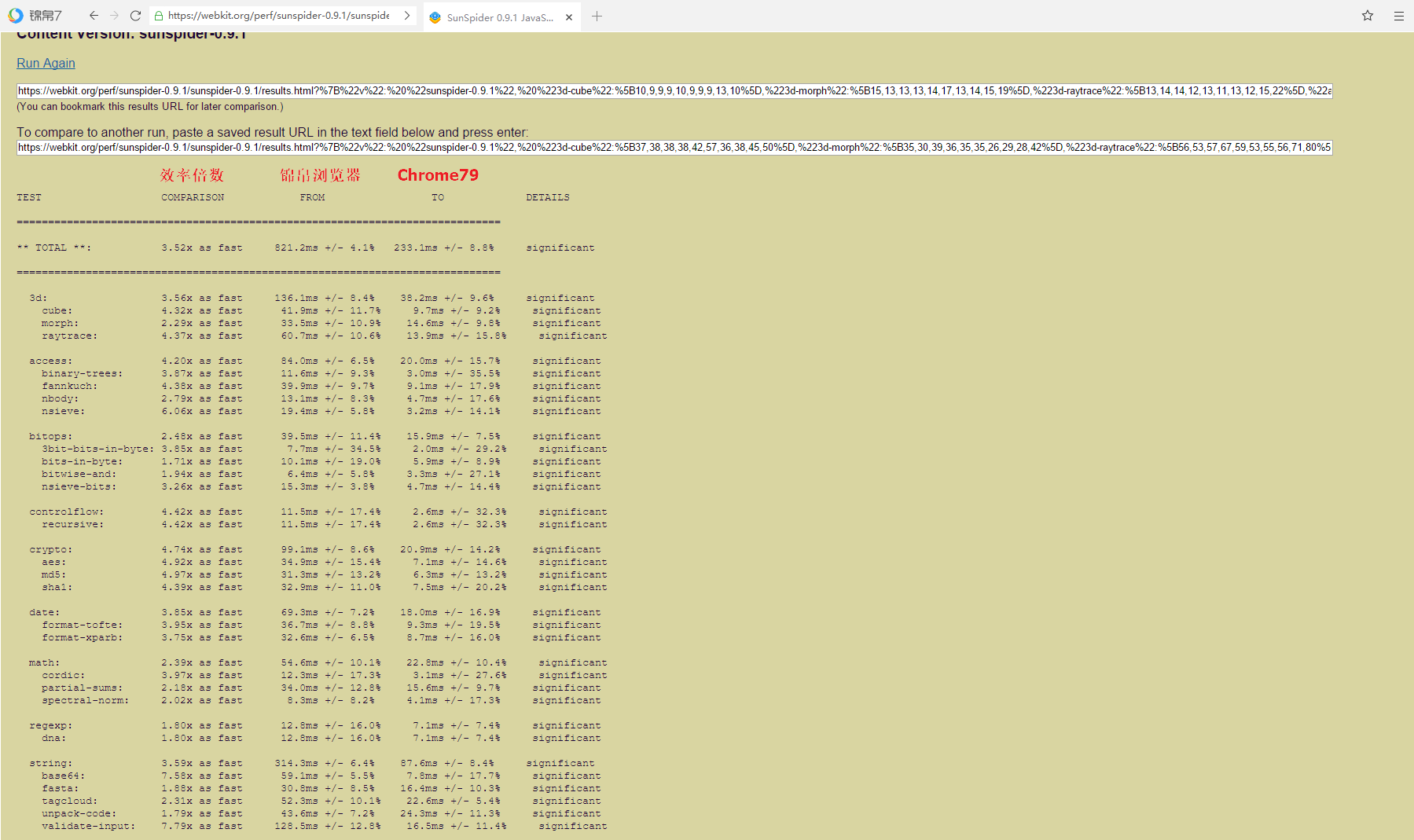
（七）对页面中可能发生大量重排重绘的元素单独触发渲染层，使用GPU分担CPU压力。（这项策略需要慎用，得着重考量以牺牲GPU占用率能否换来可期的性能优化，毕竟页面中存在太多的渲染层对与GPU而言也是一种不必要的压力，通常情况下，应该对动画元素采取硬件加速。）

锦帛浏览器VS Chrome79 VS Chrome80运行速度对比：

采取SunSpider JavaScript Benchmark 测试

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。

对比结果1：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：

对比结果2：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。

VUE官网 VUE REACR REGULAR 工具 Github

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：

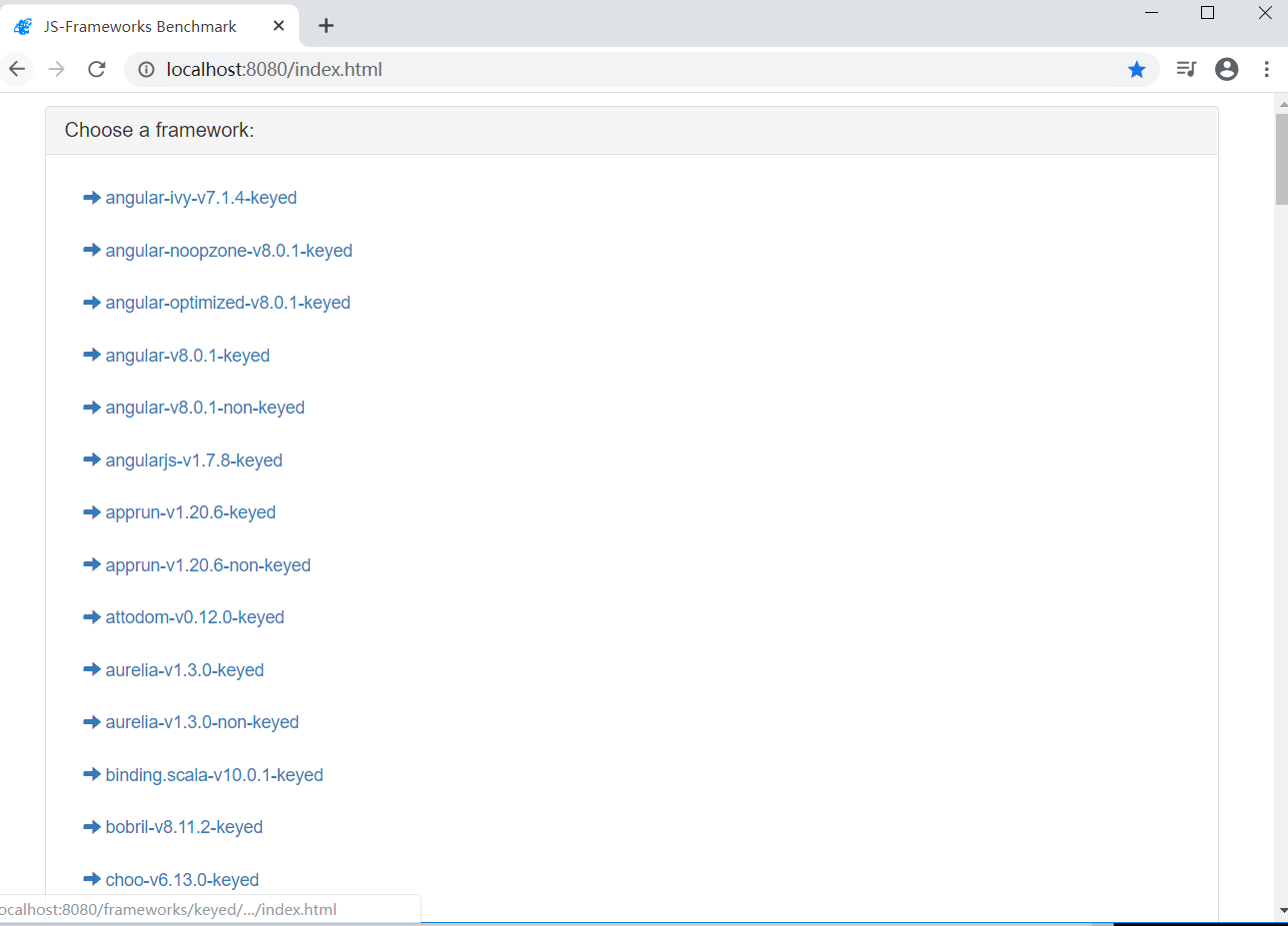
<https://github.com/krausest/js-framework-benchmark>

框架要求环境配置：

1. Nodejs >= 10.0
2. Java >= 8

启动服务：

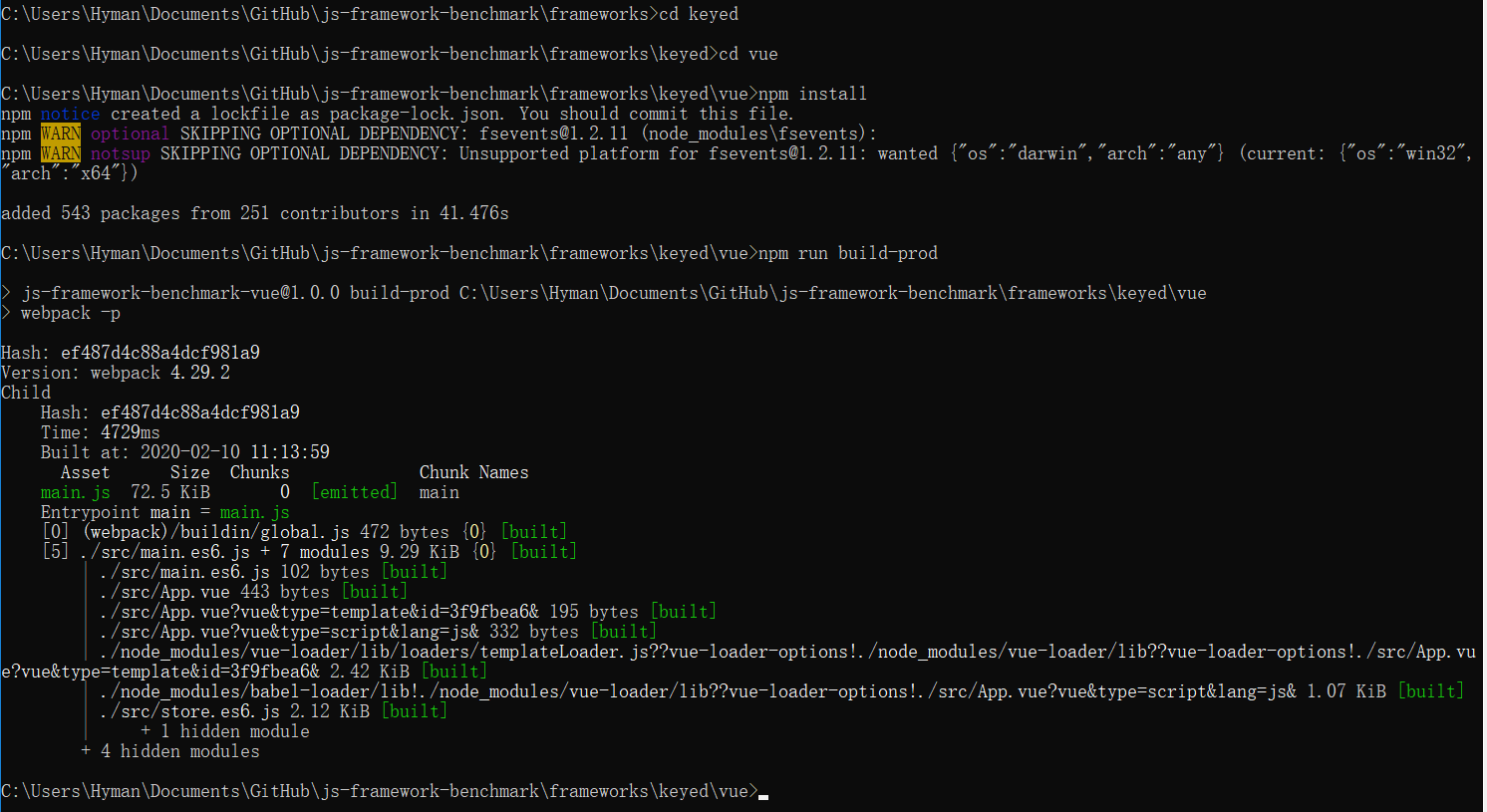
1. js-framework-benchmark服务目录下，安装依赖npm install
2. npm start
3. 打开测试页 localhost:8080/index.html



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：

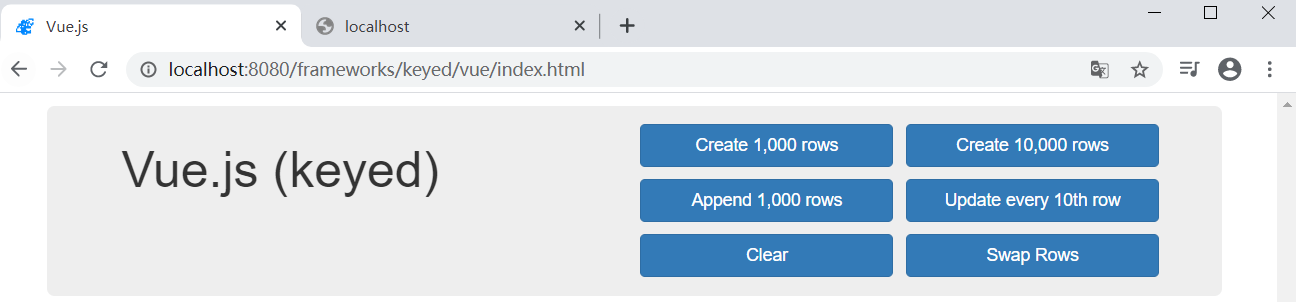
1. 进入目录 frameworks/keyed/vue
2. 安装依赖 npm install
3. 构建框架 npm run build-prod



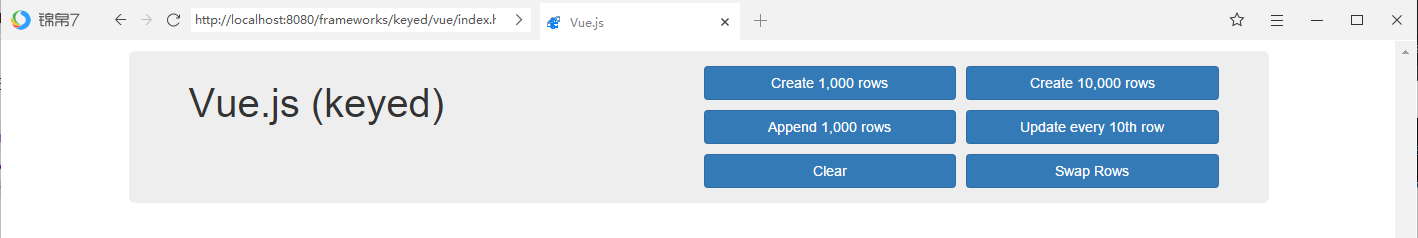
1. 在测试浏览器上，分别打开框架测试页查看是否显示正常:

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

Chrome80:



锦帛浏览器：

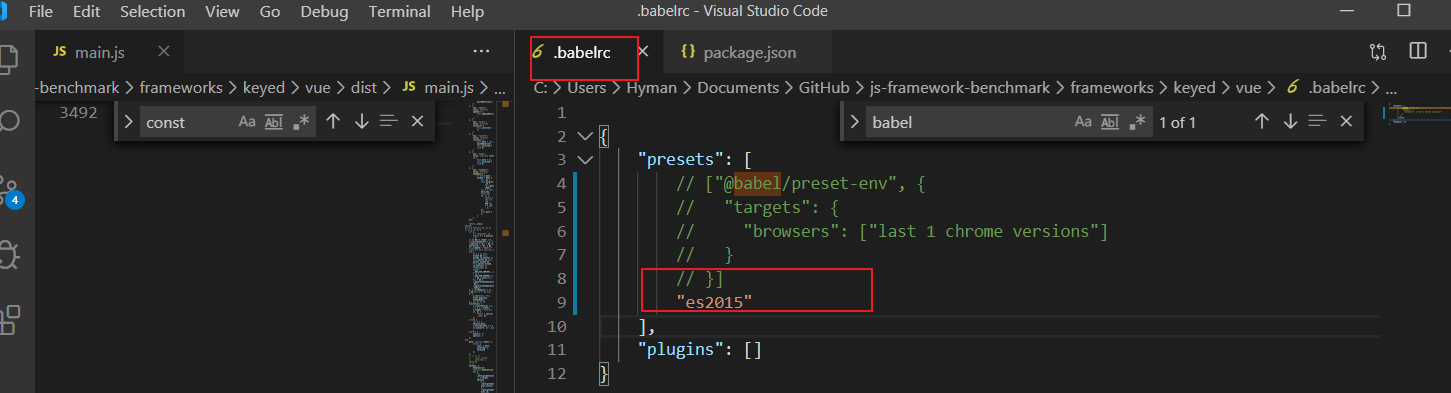


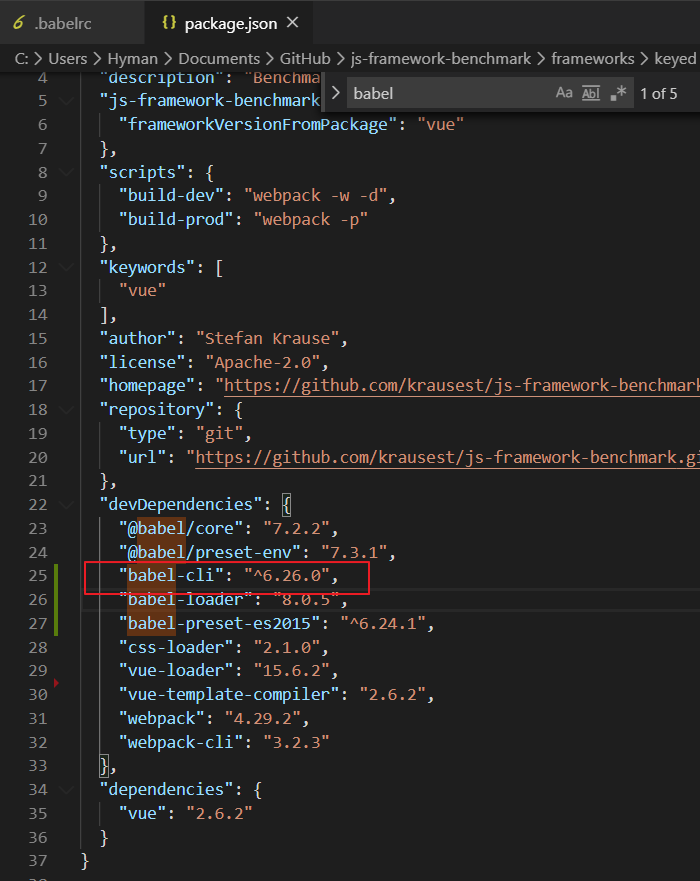
遇到的问题及解决：

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





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

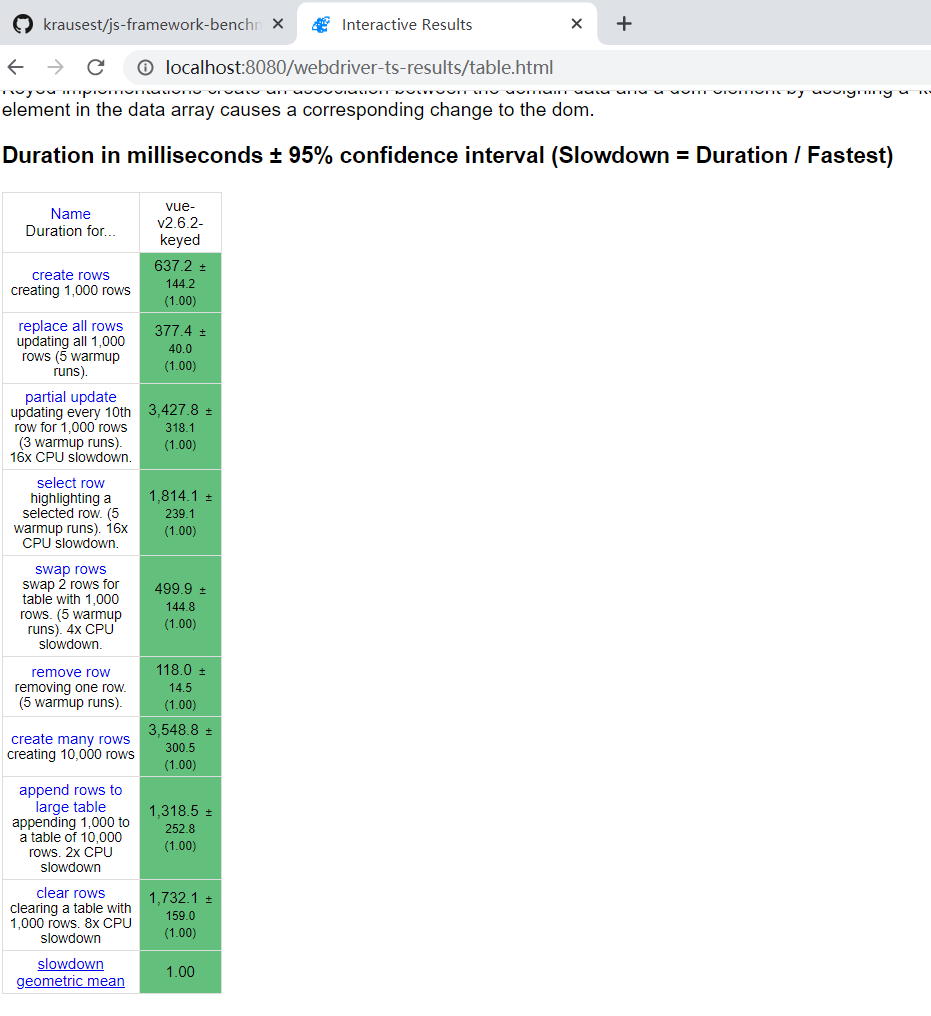
注意事项：上步启动的8080服务端口不可停止，此处测试框架调用必需。

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>



箭头函数

入参默认值

…扩展运算符