Benchpress

Webapp performance testing with WebDriver

Tobias Bosch ([tbosch@google.com](mailto:tbosch@google.com))

3/2/2015

[Motivation](#h.19yfohttlqy7)

[Prior art](#h.1l1doupsj5c4)

[Approaches that measure performance in the browser](#h.w9c4klm3kadv)

[Missing metrics](#h.ua95zd1zgq58)

[Requiring the application under test to provide special benchmark hooks](#h.u60snb7gn7y5)

[Chromium Telemetry](#h.g74xndoldo9e)

[Chromedriver 2 performance logs](#h.bw5tp1elw0ms)

[iOS driver performance logs](#h.dyz0qcgninmb)

[Design overview](#h.hvh20tq4q9v6)

[Overview](#h.8nglbsb0qz31)

[Definitions](#h.lvpp7cd07j9p)

[Runner](#h.eez4t82toaan)

[Sampler](#h.v2fbr6ptlyak)

[Metric](#h.5unqc2yg64sv)

[Validator](#h.8rliux8rynei)

[Reporter](#h.bmo6krbvbwgc)

[WebDriverAdapter](#h.uove91yxbj26)

[WebDriverExtension](#h.8pnfrifjlc9b)

[Best practises](#h.gvfke4wvzg0i)

[Use normalized environments](#h.didw16kx223)

[Use relative comparisons](#h.c82218w8c6c7)

[Assert post-commit for commit ranges](#h.7dtuj0w99x2h)

[Repeat benchmarks multiple times in a fresh window](#h.snixxhhliitf)

[Caveats](#h.geobpk6dhi6m)

[Not all browsers are supported](#h.1wo4hlr7u8ef)

# Motivation

Modern browser have similar tools for manually analyzing web app performance (e.g. Chrome DevTools Timeline, Firefox DevTools Timeline, …). They are used very often while optimizing web app performance. However, there is no standard way to use those tools in an automated way to measure web app performance, especially not across platforms.

People are already writing ui tests with WebDriver. Adding functionality to allow them to sample their interaction with the page would allow them to easily write performance tests.

# Prior art

## Approaches that measure performance in the browser

Some performance numbers can be retrieved from within the browser: e.g. execution time or memory consumption. However, this is not enough for measuring general web applications:

### Missing metrics

Calling a JS function is not always the same as a real user interaction (e.g. clicking on buttons, ...):

* the rendering time needs to be measured as well to get the overall execution time of a user interaction
* measuring the framerate while scrolling through an infinite table can help to find regressions in such performance sensitive components
* measuring forced layouts in an application would help to find the number 1 mistake users in AngularJS 1.x make in their apps in an automated way.
* garbage collection takes time and changes the results, but there is no way to know when gc happens from within the browser

### Requiring the application under test to provide special benchmark hooks

Measuring performance from inside of the browser requires the application under test to provide hooks so the runner knows when the action that should be measured is completed.

However, this does not work for applications / frameworks that the benchmark author does not control. E.g. For Angular2 we wanted to create a tree benchmark for Polymer to compare the performance against Angular2. However, Polymer does not provide a callback when it finished applying changes, as this is done through Object.observe which is called via the browser micro task queue, which does not have a done callback.

## [Chromium Telemetry](http://www.chromium.org/developers/telemetry)

Telemetry directly interfaces with the Chrome [Remote Debugging Protocol](https://developer.chrome.com/devtools/docs/debugger-protocol) to access the data of the Chrome Timeline and Chrome Tracing. However, it is tightly linked to Chrome, and tests need to be written in a special way.

## [Chromedriver 2 performance logs](https://sites.google.com/a/chromium.org/chromedriver/logging/performance-log)

Chromedriver already provides a way to output the timeline and tracing events via chromedriver. However, other browsers have not implemented this yet

## [iOS driver performance logs](https://github.com/ios-driver/ios-driver)

iOS driver already interfaces with the WebKit Remote Debug Protocol for iOS and provides performance logs as well

# Design overview

Benchpress is a tool to sample and measure interactions with a page via webdriver.

## Overview



### Definitions

* *valid sample*: a sample that represents the world that should be measured in a good way.
* *complete sample*: sample of all measure values collected so far

### Runner

* contains a default configuration
* creates a new injector for every sample call, via which all other components are created

### Sampler

* gets data from the metrics
* reports measure values immediately to the reporters
* loops until the validator is able to extract a valid sample out of the complete sample (see below).
* reports the valid sample and the complete sample to the reporters

### Metric

* gets measured values from the browser
* e.g. reads out performance logs, DOM values, JavaScript values

### Validator

* extracts a valid sample out of the complete sample of all measure values.
* e.g. wait until there are 10 samples and take them as valid sample (would include warmup time)
* e.g. wait until the regression slope for the metric 'script' through the last 10 measure values is >=0, i.e. the values for the 'script' metric are no more decreasing

### Reporter

* reports measure values, the valid sample and the complete sample to backends
* e.g. a reporter that prints to the console, a reporter that reports values into Google BigQuery, ...

### WebDriverAdapter

* abstraction over the used web driver client
* One implementation for every webdriver client
* E.g. one for selenium-webdriver Node.js module, dart async webdriver, dart sync webdriver, …

### WebDriverExtension

* implements additional methods that are standardized in the webdriver protocol using the WebDriverAdapter
* One implementation per browser
* E.g. force gc, read out performance logs in a normalized format

## Best practices

### Use normalized environments

* metrics that are dependent on the performance of the execution environment must be executed on a normalized machine
* e.g. a real mobile device whose cpu frequency is set to a fixed value
* e.g. a calibrated machine that does not run background jobs, has a fixed cpu frequency, ...

### Use relative comparisons

* relative comparisons are less likely to change over time and help to interpret the results of benchmarks
* e.g. compare an example written using a ui framework against a hand coded example and track the ratio

### Assert post-commit for commit ranges

* running benchmarks can take some time. Running them before every commit is usually too slow.
* when a regression is detected for a commit range, use bisection to find the problematic commit

### Repeat benchmarks multiple times in a fresh window

* run the same benchmark multiple times in a fresh window and then take the minimal average value of each benchmark run

### Use force gc with care

* forcing gc can skew the script execution time and gcTime numbers

### Open a new window for every test

* Browsers (e.g. chrome) might keep JIT statistics over page reloads and optimize pages differently depending on what has been loaded before

# 

# Caveats

## Not all browsers are supported

Right now, performance logs are only available for Chrome (all platforms, via chromedriver 2) and iOS Safari (via iOS driver). However, part of the benchpress effort is contributing to existing webdriver drivers to add this missing functionality (e.g. for Firefox).