Node.js Core Benchmarks

This folder contains code and data used to measure performance of different Node.js implementations and different ways of writing JavaScript run by the built-in JavaScript engine.

For a detailed guide on how to write and run benchmarks in this directory, see the guide on benchmarks.

Table of Contents

Benchmark Directories

Directory	Purpose
assert	Benchmarks for the `assert` subsystem.
buffers	Benchmarks for the `buffer` subsystem.
child_process	Benchmarks for the `child_process` subsystem.
crypto	Benchmarks for the `crypto` subsystem.
dgram	Benchmarks for the `dgram` subsystem.
domain	Benchmarks for the `domain` subsystem.
es	Benchmarks for various new ECMAScript features and their pre-ES2015 counterparts.
events	Benchmarks for the `events` subsystem.
fixtures	Benchmarks fixtures used in various benchmarks throughout the benchmark suite.
fs	Benchmarks for the `fs` subsystem.
http	Benchmarks for the `http` subsystem.
http2	Benchmarks for the `http2` subsystem.
misc	Miscellaneous benchmarks and benchmarks for shared internal modules.
module	Benchmarks for the `module` subsystem.
net	Benchmarks for the `net` subsystem.
path	Benchmarks for the `path` subsystem.
perf_hooks	Benchmarks for the `perf_hooks` subsystem.
process	Benchmarks for the `process` subsystem.
querystring	Benchmarks for the `querystring` subsystem.
streams	Benchmarks for the `streams` subsystem.
string_decoder	Benchmarks for the `string_decoder` subsystem.

timers	Benchmarks for the `timers` subsystem, including `setTimeout`, `setInterval`, .etc.
tls	Benchmarks for the `tls` subsystem.
url	Benchmarks for the `url` subsystem, including the legacy `url` implementation and the WHATWG URL implementation.
util	Benchmarks for the `util` subsystem.
vm	Benchmarks for the `vm` subsystem.

Other Top-level files

The top-level files include common dependencies of the benchmarks and the tools for launching benchmarks and visualizing their output. The actual benchmark scripts should be placed in their corresponding directories.

- `_benchmark_progress.js` : implements the progress bar displayed when running `compare.js`
- `_cli.js` : parses the command line arguments passed to `compare.js` , `run.js` and `scatter.js`
- `_cli.R` : parses the command line arguments passed to `compare.R`
- `_http-benchmarkers.js` : selects and runs external tools for benchmarking the `http` subsystem.
- `common.js` : see Common API.
- `compare.js` : command line tool for comparing performance between different Node.js binaries.
- `compare.R` : R script for statistically analyzing the output of `compare.js`
- `run.js` : command line tool for running individual benchmark suite(s).
- `scatter.js` : command line tool for comparing the performance between different parameters in benchmark configurations, for example to analyze the time complexity.
- `scatter.R` : R script for visualizing the output of `scatter.js` with scatter plots.

Common API

The common.js module is used by benchmarks for consistency across repeated tasks. It has a number of helpful functions and properties to help with writing benchmarks.

`createBenchmark(fn, configs[, options])`

See the guide on writing benchmarks.

`default_http_benchmarker`

The default benchmarker used to run HTTP benchmarks. See the guide on writing HTTP benchmarks.

`PORT`

The default port used to run HTTP benchmarks. See the guide on writing HTTP benchmarks.

`sendResult(data)`

Used in special benchmarks that can't use `createBenchmark` and the object it returns to accomplish what they need. This function reports timing data to the parent process (usually created by running `compare.js` , `run.js` or `scatter.js`).