/\*\*

\* This module provides an implementation of a subset of the W3C [Web Performance APIs](https://w3c.github.io/perf-timing-primer/) as well as additional APIs for

\* Node.js-specific performance measurements.

\*

\* Node.js supports the following [Web Performance APIs](https://w3c.github.io/perf-timing-primer/):

\*

\* \* [High Resolution Time](https://www.w3.org/TR/hr-time-2)

\* \* [Performance Timeline](https://w3c.github.io/performance-timeline/)

\* \* [User Timing](https://www.w3.org/TR/user-timing/)

\*

\* ```js

\* const { PerformanceObserver, performance } = require('perf\_hooks');

\*

\* const obs = new PerformanceObserver((items) => {

\* console.log(items.getEntries()[0].duration);

\* performance.clearMarks();

\* });

\* obs.observe({ type: 'measure' });

\* performance.measure('Start to Now');

\*

\* performance.mark('A');

\* doSomeLongRunningProcess(() => {

\* performance.measure('A to Now', 'A');

\*

\* performance.mark('B');

\* performance.measure('A to B', 'A', 'B');

\* });

\* ```

\* @see [source](https://github.com/nodejs/node/blob/v17.0.0/lib/perf\_hooks.js)

\*/

declare module 'perf\_hooks' {

import { AsyncResource } from 'node:async\_hooks';

type EntryType = 'node' | 'mark' | 'measure' | 'gc' | 'function' | 'http2' | 'http';

interface NodeGCPerformanceDetail {

/\*\*

\* When `performanceEntry.entryType` is equal to 'gc', `the performance.kind` property identifies

\* the type of garbage collection operation that occurred.

\* See perf\_hooks.constants for valid values.

\*/

readonly kind?: number | undefined;

/\*\*

\* When `performanceEntry.entryType` is equal to 'gc', the `performance.flags`

\* property contains additional information about garbage collection operation.

\* See perf\_hooks.constants for valid values.

\*/

readonly flags?: number | undefined;

}

/\*\*

\* @since v8.5.0

\*/

class PerformanceEntry {

protected constructor();

/\*\*

\* The total number of milliseconds elapsed for this entry. This value will not

\* be meaningful for all Performance Entry types.

\* @since v8.5.0

\*/

readonly duration: number;

/\*\*

\* The name of the performance entry.

\* @since v8.5.0

\*/

readonly name: string;

/\*\*

\* The high resolution millisecond timestamp marking the starting time of the

\* Performance Entry.

\* @since v8.5.0

\*/

readonly startTime: number;

/\*\*

\* The type of the performance entry. It may be one of:

\*

\* \* `'node'` (Node.js only)

\* \* `'mark'` (available on the Web)

\* \* `'measure'` (available on the Web)

\* \* `'gc'` (Node.js only)

\* \* `'function'` (Node.js only)

\* \* `'http2'` (Node.js only)

\* \* `'http'` (Node.js only)

\* @since v8.5.0

\*/

readonly entryType: EntryType;

/\*\*

\* Additional detail specific to the `entryType`.

\* @since v16.0.0

\*/

readonly detail?: NodeGCPerformanceDetail | unknown | undefined; // TODO: Narrow this based on entry type.

}

/\*\*

\* \_This property is an extension by Node.js. It is not available in Web browsers.\_

\*

\* Provides timing details for Node.js itself. The constructor of this class

\* is not exposed to users.

\* @since v8.5.0

\*/

class PerformanceNodeTiming extends PerformanceEntry {

/\*\*

\* The high resolution millisecond timestamp at which the Node.js process

\* completed bootstrapping. If bootstrapping has not yet finished, the property

\* has the value of -1.

\* @since v8.5.0

\*/

readonly bootstrapComplete: number;

/\*\*

\* The high resolution millisecond timestamp at which the Node.js environment was

\* initialized.

\* @since v8.5.0

\*/

readonly environment: number;

/\*\*

\* The high resolution millisecond timestamp of the amount of time the event loop

\* has been idle within the event loop's event provider (e.g. `epoll\_wait`). This

\* does not take CPU usage into consideration. If the event loop has not yet

\* started (e.g., in the first tick of the main script), the property has the

\* value of 0.

\* @since v14.10.0, v12.19.0

\*/

readonly idleTime: number;

/\*\*

\* The high resolution millisecond timestamp at which the Node.js event loop

\* exited. If the event loop has not yet exited, the property has the value of -1\.

\* It can only have a value of not -1 in a handler of the `'exit'` event.

\* @since v8.5.0

\*/

readonly loopExit: number;

/\*\*

\* The high resolution millisecond timestamp at which the Node.js event loop

\* started. If the event loop has not yet started (e.g., in the first tick of the

\* main script), the property has the value of -1.

\* @since v8.5.0

\*/

readonly loopStart: number;

/\*\*

\* The high resolution millisecond timestamp at which the V8 platform was

\* initialized.

\* @since v8.5.0

\*/

readonly v8Start: number;

}

interface EventLoopUtilization {

idle: number;

active: number;

utilization: number;

}

/\*\*

\* @param util1 The result of a previous call to eventLoopUtilization()

\* @param util2 The result of a previous call to eventLoopUtilization() prior to util1

\*/

type EventLoopUtilityFunction = (util1?: EventLoopUtilization, util2?: EventLoopUtilization) => EventLoopUtilization;

interface MarkOptions {

/\*\*

\* Additional optional detail to include with the mark.

\*/

detail?: unknown | undefined;

/\*\*

\* An optional timestamp to be used as the mark time.

\* @default `performance.now()`.

\*/

startTime?: number | undefined;

}

interface MeasureOptions {

/\*\*

\* Additional optional detail to include with the mark.

\*/

detail?: unknown | undefined;

/\*\*

\* Duration between start and end times.

\*/

duration?: number | undefined;

/\*\*

\* Timestamp to be used as the end time, or a string identifying a previously recorded mark.

\*/

end?: number | string | undefined;

/\*\*

\* Timestamp to be used as the start time, or a string identifying a previously recorded mark.

\*/

start?: number | string | undefined;

}

interface TimerifyOptions {

/\*\*

\* A histogram object created using

\* `perf\_hooks.createHistogram()` that will record runtime durations in

\* nanoseconds.

\*/

histogram?: RecordableHistogram | undefined;

}

interface Performance {

/\*\*

\* If name is not provided, removes all PerformanceMark objects from the Performance Timeline.

\* If name is provided, removes only the named mark.

\* @param name

\*/

clearMarks(name?: string): void;

/\*\*

\* Creates a new PerformanceMark entry in the Performance Timeline.

\* A PerformanceMark is a subclass of PerformanceEntry whose performanceEntry.entryType is always 'mark',

\* and whose performanceEntry.duration is always 0.

\* Performance marks are used to mark specific significant moments in the Performance Timeline.

\* @param name

\*/

mark(name?: string, options?: MarkOptions): void;

/\*\*

\* Creates a new PerformanceMeasure entry in the Performance Timeline.

\* A PerformanceMeasure is a subclass of PerformanceEntry whose performanceEntry.entryType is always 'measure',

\* and whose performanceEntry.duration measures the number of milliseconds elapsed since startMark and endMark.

\*

\* The startMark argument may identify any existing PerformanceMark in the the Performance Timeline, or may identify

\* any of the timestamp properties provided by the PerformanceNodeTiming class. If the named startMark does not exist,

\* then startMark is set to timeOrigin by default.

\*

\* The endMark argument must identify any existing PerformanceMark in the the Performance Timeline or any of the timestamp

\* properties provided by the PerformanceNodeTiming class. If the named endMark does not exist, an error will be thrown.

\* @param name

\* @param startMark

\* @param endMark

\*/

measure(name: string, startMark?: string, endMark?: string): void;

measure(name: string, options: MeasureOptions): void;

/\*\*

\* An instance of the PerformanceNodeTiming class that provides performance metrics for specific Node.js operational milestones.

\*/

readonly nodeTiming: PerformanceNodeTiming;

/\*\*

\* @return the current high resolution millisecond timestamp

\*/

now(): number;

/\*\*

\* The timeOrigin specifies the high resolution millisecond timestamp from which all performance metric durations are measured.

\*/

readonly timeOrigin: number;

/\*\*

\* Wraps a function within a new function that measures the running time of the wrapped function.

\* A PerformanceObserver must be subscribed to the 'function' event type in order for the timing details to be accessed.

\* @param fn

\*/

timerify<T extends (...params: any[]) => any>(fn: T, options?: TimerifyOptions): T;

/\*\*

\* eventLoopUtilization is similar to CPU utilization except that it is calculated using high precision wall-clock time.

\* It represents the percentage of time the event loop has spent outside the event loop's event provider (e.g. epoll\_wait).

\* No other CPU idle time is taken into consideration.

\*/

eventLoopUtilization: EventLoopUtilityFunction;

}

interface PerformanceObserverEntryList {

/\*\*

\* Returns a list of `PerformanceEntry` objects in chronological order

\* with respect to `performanceEntry.startTime`.

\*

\* ```js

\* const {

\* performance,

\* PerformanceObserver

\* } = require('perf\_hooks');

\*

\* const obs = new PerformanceObserver((perfObserverList, observer) => {

\* console.log(perfObserverList.getEntries());

\*

\* \* [

\* \* PerformanceEntry {

\* \* name: 'test',

\* \* entryType: 'mark',

\* \* startTime: 81.465639,

\* \* duration: 0

\* \* },

\* \* PerformanceEntry {

\* \* name: 'meow',

\* \* entryType: 'mark',

\* \* startTime: 81.860064,

\* \* duration: 0

\* \* }

\* \* ]

\*

\* observer.disconnect();

\* });

\* obs.observe({ type: 'mark' });

\*

\* performance.mark('test');

\* performance.mark('meow');

\* ```

\* @since v8.5.0

\*/

getEntries(): PerformanceEntry[];

/\*\*

\* Returns a list of `PerformanceEntry` objects in chronological order

\* with respect to `performanceEntry.startTime` whose `performanceEntry.name` is

\* equal to `name`, and optionally, whose `performanceEntry.entryType` is equal to`type`.

\*

\* ```js

\* const {

\* performance,

\* PerformanceObserver

\* } = require('perf\_hooks');

\*

\* const obs = new PerformanceObserver((perfObserverList, observer) => {

\* console.log(perfObserverList.getEntriesByName('meow'));

\*

\* \* [

\* \* PerformanceEntry {

\* \* name: 'meow',

\* \* entryType: 'mark',

\* \* startTime: 98.545991,

\* \* duration: 0

\* \* }

\* \* ]

\*

\* console.log(perfObserverList.getEntriesByName('nope')); // []

\*

\* console.log(perfObserverList.getEntriesByName('test', 'mark'));

\*

\* \* [

\* \* PerformanceEntry {

\* \* name: 'test',

\* \* entryType: 'mark',

\* \* startTime: 63.518931,

\* \* duration: 0

\* \* }

\* \* ]

\*

\* console.log(perfObserverList.getEntriesByName('test', 'measure')); // []

\* observer.disconnect();

\* });

\* obs.observe({ entryTypes: ['mark', 'measure'] });

\*

\* performance.mark('test');

\* performance.mark('meow');

\* ```

\* @since v8.5.0

\*/

getEntriesByName(name: string, type?: EntryType): PerformanceEntry[];

/\*\*

\* Returns a list of `PerformanceEntry` objects in chronological order

\* with respect to `performanceEntry.startTime` whose `performanceEntry.entryType`is equal to `type`.

\*

\* ```js

\* const {

\* performance,

\* PerformanceObserver

\* } = require('perf\_hooks');

\*

\* const obs = new PerformanceObserver((perfObserverList, observer) => {

\* console.log(perfObserverList.getEntriesByType('mark'));

\*

\* \* [

\* \* PerformanceEntry {

\* \* name: 'test',

\* \* entryType: 'mark',

\* \* startTime: 55.897834,

\* \* duration: 0

\* \* },

\* \* PerformanceEntry {

\* \* name: 'meow',

\* \* entryType: 'mark',

\* \* startTime: 56.350146,

\* \* duration: 0

\* \* }

\* \* ]

\*

\* observer.disconnect();

\* });

\* obs.observe({ type: 'mark' });

\*

\* performance.mark('test');

\* performance.mark('meow');

\* ```

\* @since v8.5.0

\*/

getEntriesByType(type: EntryType): PerformanceEntry[];

}

type PerformanceObserverCallback = (list: PerformanceObserverEntryList, observer: PerformanceObserver) => void;

class PerformanceObserver extends AsyncResource {

constructor(callback: PerformanceObserverCallback);

/\*\*

\* Disconnects the `PerformanceObserver` instance from all notifications.

\* @since v8.5.0

\*/

disconnect(): void;

/\*\*

\* Subscribes the `PerformanceObserver` instance to notifications of new `PerformanceEntry` instances identified either by `options.entryTypes`or `options.type`:

\*

\* ```js

\* const {

\* performance,

\* PerformanceObserver

\* } = require('perf\_hooks');

\*

\* const obs = new PerformanceObserver((list, observer) => {

\* // Called three times synchronously. `list` contains one item.

\* });

\* obs.observe({ type: 'mark' });

\*

\* for (let n = 0; n < 3; n++)

\* performance.mark(`test${n}`);

\* ```

\* @since v8.5.0

\*/

observe(

options:

| {

entryTypes: ReadonlyArray<EntryType>;

buffered?: boolean | undefined;

}

| {

type: EntryType;

buffered?: boolean | undefined;

}

): void;

}

namespace constants {

const NODE\_PERFORMANCE\_GC\_MAJOR: number;

const NODE\_PERFORMANCE\_GC\_MINOR: number;

const NODE\_PERFORMANCE\_GC\_INCREMENTAL: number;

const NODE\_PERFORMANCE\_GC\_WEAKCB: number;

const NODE\_PERFORMANCE\_GC\_FLAGS\_NO: number;

const NODE\_PERFORMANCE\_GC\_FLAGS\_CONSTRUCT\_RETAINED: number;

const NODE\_PERFORMANCE\_GC\_FLAGS\_FORCED: number;

const NODE\_PERFORMANCE\_GC\_FLAGS\_SYNCHRONOUS\_PHANTOM\_PROCESSING: number;

const NODE\_PERFORMANCE\_GC\_FLAGS\_ALL\_AVAILABLE\_GARBAGE: number;

const NODE\_PERFORMANCE\_GC\_FLAGS\_ALL\_EXTERNAL\_MEMORY: number;

const NODE\_PERFORMANCE\_GC\_FLAGS\_SCHEDULE\_IDLE: number;

}

const performance: Performance;

interface EventLoopMonitorOptions {

/\*\*

\* The sampling rate in milliseconds.

\* Must be greater than zero.

\* @default 10

\*/

resolution?: number | undefined;

}

interface Histogram {

/\*\*

\* Returns a `Map` object detailing the accumulated percentile distribution.

\* @since v11.10.0

\*/

readonly percentiles: Map<number, number>;

/\*\*

\* The number of times the event loop delay exceeded the maximum 1 hour event

\* loop delay threshold.

\* @since v11.10.0

\*/

readonly exceeds: number;

/\*\*

\* The minimum recorded event loop delay.

\* @since v11.10.0

\*/

readonly min: number;

/\*\*

\* The maximum recorded event loop delay.

\* @since v11.10.0

\*/

readonly max: number;

/\*\*

\* The mean of the recorded event loop delays.

\* @since v11.10.0

\*/

readonly mean: number;

/\*\*

\* The standard deviation of the recorded event loop delays.

\* @since v11.10.0

\*/

readonly stddev: number;

/\*\*

\* Resets the collected histogram data.

\* @since v11.10.0

\*/

reset(): void;

/\*\*

\* Returns the value at the given percentile.

\* @since v11.10.0

\* @param percentile A percentile value in the range (0, 100].

\*/

percentile(percentile: number): number;

}

interface IntervalHistogram extends Histogram {

/\*\*

\* Enables the update interval timer. Returns `true` if the timer was

\* started, `false` if it was already started.

\* @since v11.10.0

\*/

enable(): boolean;

/\*\*

\* Disables the update interval timer. Returns `true` if the timer was

\* stopped, `false` if it was already stopped.

\* @since v11.10.0

\*/

disable(): boolean;

}

interface RecordableHistogram extends Histogram {

/\*\*

\* @since v15.9.0, v14.18.0

\* @param val The amount to record in the histogram.

\*/

record(val: number | bigint): void;

/\*\*

\* Calculates the amount of time (in nanoseconds) that has passed since the

\* previous call to `recordDelta()` and records that amount in the histogram.

\*

\* ## Examples

\* @since v15.9.0, v14.18.0

\*/

recordDelta(): void;

}

/\*\*

\* \_This property is an extension by Node.js. It is not available in Web browsers.\_

\*

\* Creates an `IntervalHistogram` object that samples and reports the event loop

\* delay over time. The delays will be reported in nanoseconds.

\*

\* Using a timer to detect approximate event loop delay works because the

\* execution of timers is tied specifically to the lifecycle of the libuv

\* event loop. That is, a delay in the loop will cause a delay in the execution

\* of the timer, and those delays are specifically what this API is intended to

\* detect.

\*

\* ```js

\* const { monitorEventLoopDelay } = require('perf\_hooks');

\* const h = monitorEventLoopDelay({ resolution: 20 });

\* h.enable();

\* // Do something.

\* h.disable();

\* console.log(h.min);

\* console.log(h.max);

\* console.log(h.mean);

\* console.log(h.stddev);

\* console.log(h.percentiles);

\* console.log(h.percentile(50));

\* console.log(h.percentile(99));

\* ```

\* @since v11.10.0

\*/

function monitorEventLoopDelay(options?: EventLoopMonitorOptions): IntervalHistogram;

interface CreateHistogramOptions {

/\*\*

\* The minimum recordable value. Must be an integer value greater than 0.

\* @default 1

\*/

min?: number | bigint | undefined;

/\*\*

\* The maximum recordable value. Must be an integer value greater than min.

\* @default Number.MAX\_SAFE\_INTEGER

\*/

max?: number | bigint | undefined;

/\*\*

\* The number of accuracy digits. Must be a number between 1 and 5.

\* @default 3

\*/

figures?: number | undefined;

}

/\*\*

\* Returns a `RecordableHistogram`.

\* @since v15.9.0, v14.18.0

\*/

function createHistogram(options?: CreateHistogramOptions): RecordableHistogram;

}

declare module 'node:perf\_hooks' {

export \* from 'perf\_hooks';

}