Node.js Test Runner

Why is it game-changing?

$whoami

Senior Software engineer_

Klarna.

Google Developers Experts 2019
Web Technologies GDE

MVP Microsoft® Most Valuable Professional

docker CAPTAINS

https://{twitter,instagram,github,youtube,linkedin}.lsantos.dev

What is my objective here?

What is my objective here?

make you ditch Jest

What is my objective here?

make you ditch Jest

yes, it's that simple...

and how am I going to do that?

and how am I going to do that?

by showing you something (possibly) better

Why "possibly"?

Why "possibly"?

because the Node.js test runner is still being improved

What's bad about current runners?

What's bad about current runners?

What's bad about current runners?

1. They are difficult to configure, especially for TypeScript

What's bad about current runners?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow

What's bad about current runners?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project

What's bad about current runners?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project
4. Test runners seem to not interoperate well with each other, making it hard to switch

What's bad about current runners?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project
4. Test runners seem to not interoperate well with each other, making it hard to switch
5. Most of them are pretty opinionated, some of them are just old

What's bad about current runners?

1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project
4. Test runners seem to not interoperate well with each other, making it hard to switch
5. Most of them are pretty opinionated, some of them are just old
6. Want TypeScript, but don't want to use ts-jest? Good luck!

What's bad about current runners?

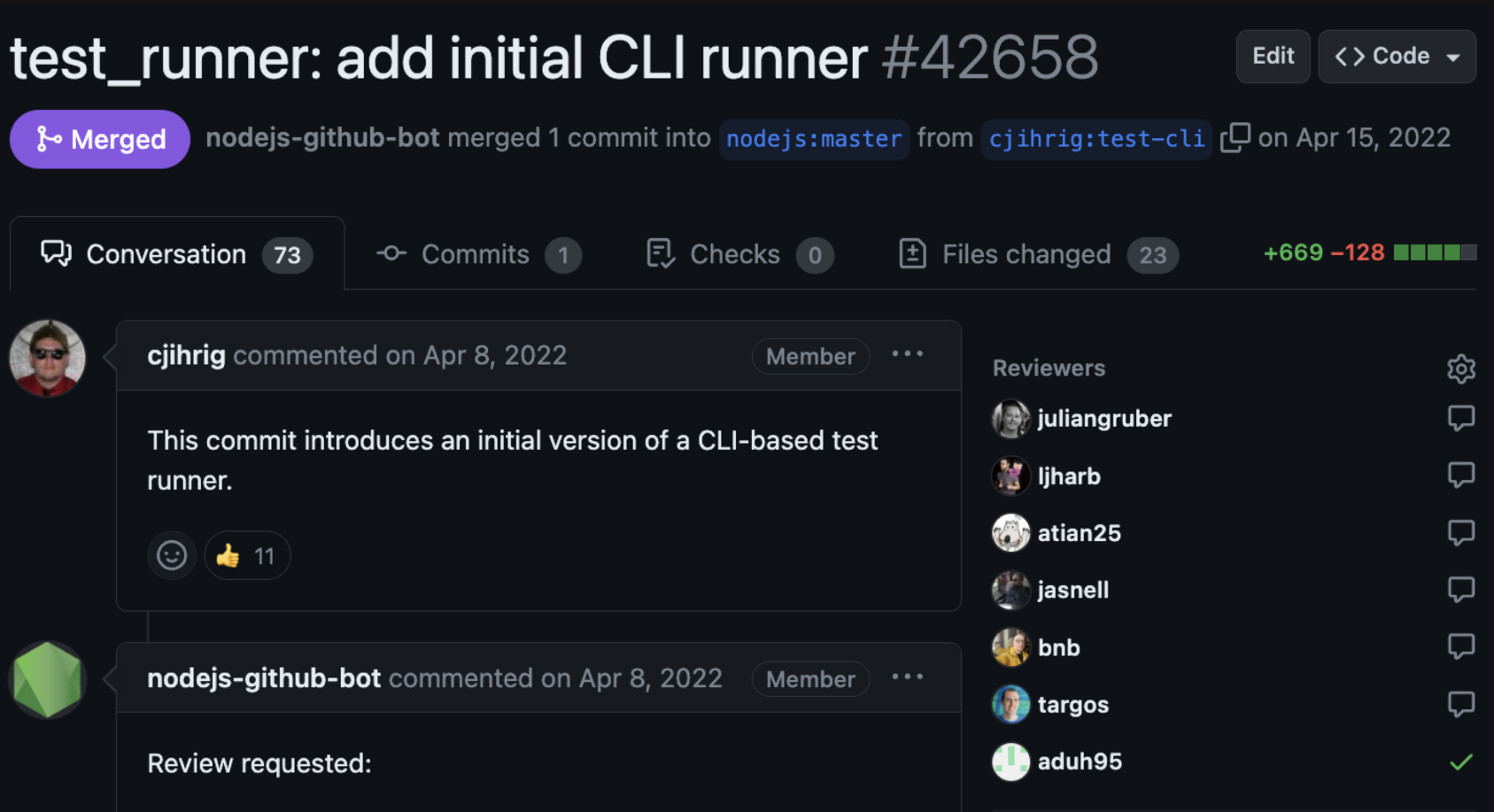
1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project
4. Test runners seem to not interoperate well with each other, making it hard to switch
5. Most of them are pretty opinionated, some of them are just old
6. Want TypeScript, but don't want to use ts-jest? Good luck!
    1. Otherwise, **YAL**

What's bad about current runners?

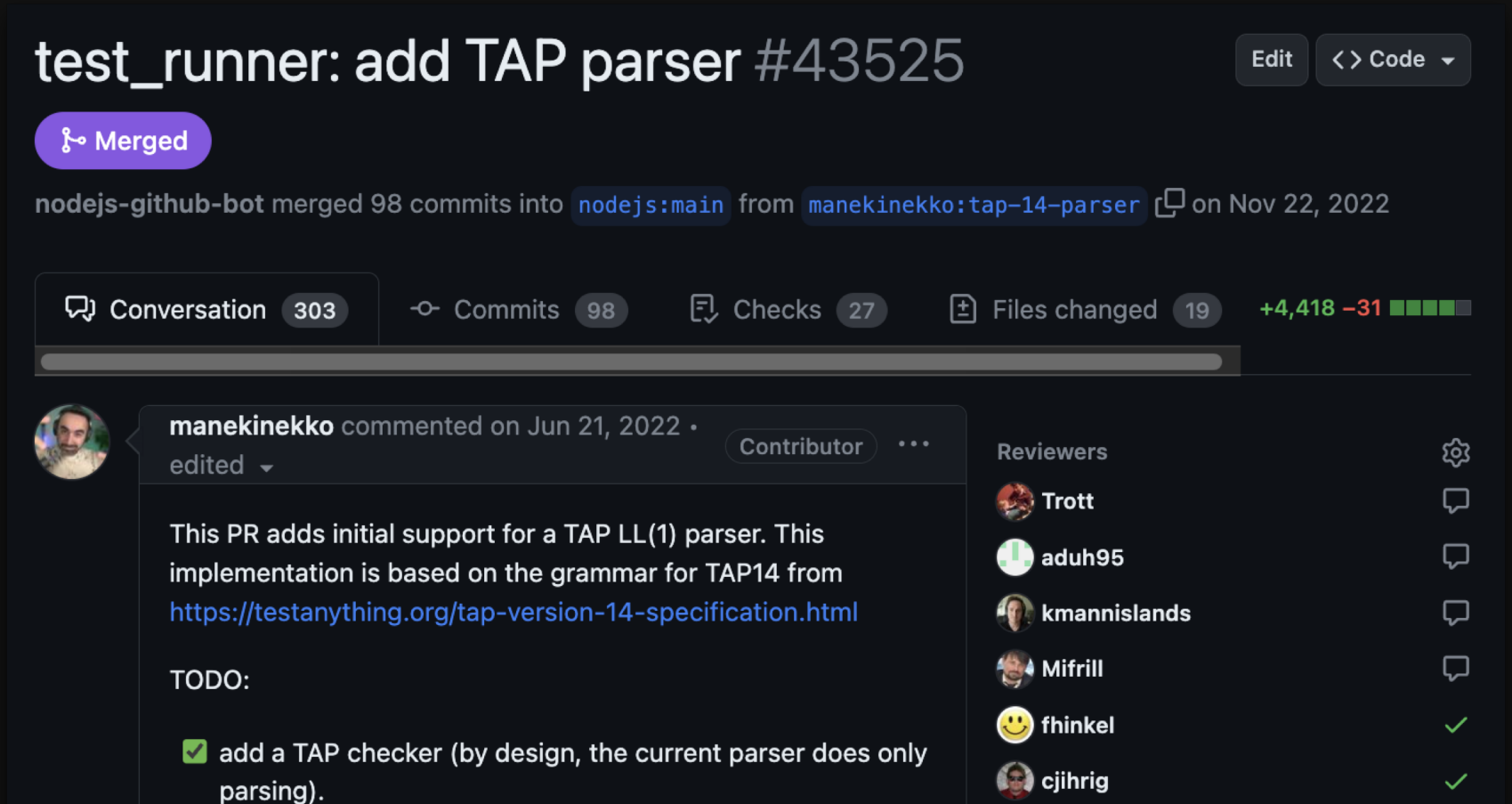
1. They are difficult to configure, especially for TypeScript
2. (Some) Are slow
3. **YAL** (Yet Another Library) on top of your project
4. Test runners seem to not interoperate well with each other, making it hard to switch
5. Most of them are pretty opinionated, some of them are just old
6. Want TypeScript, but don't want to use ts-jest? Good luck!
    1. Otherwise, **YAL**
7. There are SO MANY of them, it's hard to choose

Enters the Node.js Test Runner_

It's not that new



Improvements came soon after



It was soon stable

Request: mark test runner stable in Node 20.0.0 #46642

Closed · 19 comments · Fixed by #46983

cjihrig commented on Feb 13, 2023 — Contributor

What is the problem this feature will solve?

Having a stable test runner in core. People have been asking for it, and I feel the majority of the test runner is ready.

What is the feature you are proposing to solve the problem?

I would like to stabilize the test runner for the 20.0.0 release. This currently would not include code coverage, which is behind a separate flag.

What alternatives have you considered?

Not stabilizing the test runner.

Why is it different?

Why is it different?

- It’s built-in, so no **YAL**

Why is it different?

- It’s built-in, so no **YAL**
- It’s fast

Why is it different?

- It’s built-in, so no **YAL**
- It’s fast
- No configuration needed, integrates seamlessly with Node.js

Why is it different?

- It’s built-in, so no **YAL**
- It’s fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library

Why is it different?

- It’s built-in, so no **YAL**
- It’s fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it’s easy to integrate with other runners

Why is it different?

- It’s built-in, so no **YAL**
- It’s fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it’s easy to integrate with other runners
- Always use up to date Node.js features

Why is it different?

- It’s built-in, so no **YAL**
- It’s fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it’s easy to integrate with other runners
- Always use up to date Node.js features
- Consistency across projects

Why is it different?

- It's built-in, so no **YAL**
- It's fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it's easy to integrate with other runners
- Always use up to date Node.js features
- Consistency across projects
- No polyfills needed

Why is it different?

- It's built-in, so no **YAL**
- It's fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it's easy to integrate with other runners
- Always use up to date Node.js features
- Consistency across projects
- No polyfills needed
- Experimental support for code coverage and other reporters

Why is it different?

- It's built-in, so no **YAL**
- It's fast
- No configuration needed, integrates seamlessly with Node.js
- Can use native assertions, or any other assertion library
- Outputs TAP, so it's easy to integrate with other runners
- Always use up to date Node.js features
- Consistency across projects
- No polyfills needed
- Experimental support for code coverage and other reporters
- Supports TS out of the box

Getting started

You don't need anything to get started. Import `node:test` and `node:assert` and you are good to go!

```
// @filename: index.js
export function sum(a, b) {
  if (typeof a !== 'number' || typeof b !== 'number') {
    throw new Error('Invalid input')
  }
  return a + b
}

// @filename: test.js
import { describe, it } from 'node:test'
import assert from 'node:assert'
import { sum } from './index.js'

describe('sum', () => {
  it('should add two numbers', () => {
    assert.strictEqual(sum(1, 2), 3)
  })
})

```

Getting started

You don't need anything to get started. Import `node:test` and `node:assert` and you are good to go!

```
// @filename: index.js
export function sum(a, b) {
  if (typeof a !== 'number' || typeof b !== 'number') {
    throw new Error('Invalid input')
  }
  return a + b
}

// @filename: test.js
import { describe, it } from 'node:test'
import assert from 'node:assert'
import { sum } from './index.js'

describe('sum', () => {
  it('should add two numbers', () => {
    assert.strictEqual(sum(1, 2), 3)
  })
})

```

Getting started

You don't need anything to get started. Import `node:test` and `node:assert` and you are good to go!

```
// @filename: index.js
export function sum(a, b) {
  if (typeof a !== 'number' || typeof b !== 'number') {
    throw new Error('Invalid input')
  }
  return a + b
}

// @filename: test.js
import { describe, it } from 'node:test'
import assert from 'node:assert'
import { sum } from './index.js'

describe('sum', () => {
  it('should add two numbers', () => {
    assert.strictEqual(sum(1, 2), 3)
  })
})

```

Getting started

You don't need anything to get started. Import `node:test` and `node:assert` and you are good to go!

```
// @filename: index.js
export function sum(a, b) {
  if (typeof a !== 'number' || typeof b !== 'number') {
    throw new Error('Invalid input')
  }
  return a + b
}

// @filename: test.js
import { describe, it } from 'node:test'
import assert from 'node:assert'
import { sum } from './index.js'

describe('sum', () => {
  it('should add two numbers', () => {
    assert.strictEqual(sum(1, 2), 3)
  })
})

```

Let's break it down…

Writing tests_

Writing tests

```
import { describe, it } from 'node:test'
import assert from 'node:assert'
import { sum } from './index.js'

describe('sum', () => {
  it('should add two numbers', () => {
    assert.strictEqual(sum(1, 2), 3)
  })
})

```

Writing tests

```
import { describe, it } from 'node:test'
import assert from 'node:assert'
import { sum } from './index.js'

describe('sum', () => {
  it('should add two numbers', () => {
    assert.strictEqual(sum(1, 2), 3)
  })
})

```

Writing tests

```
import { describe, it } from 'node:test'
import assert from 'node:assert'
import { sum } from './index.js'

describe('sum', () => {
  it('should add two numbers', () => {
    assert.strictEqual(sum(1, 2), 3)
  })
})

```

Writing tests

```
import { describe, it } from 'node:test'
import assert from 'node:assert'
import { sum } from './index.js'

describe('sum', () ⇒ {
  it('should add two numbers', () ⇒ {
    assert.strictEqual(sum(1, 2), 3)
  })
})

```

Writing tests

```
import { describe, it } from 'node:test'
import assert from 'node:assert'
import { sum } from './index.js'

describe('sum', () => {
  it('should add two numbers', () => {
    assert.strictEqual(sum(1, 2), 3)
  })
})

```

Or you can use the `test` function

```
import test from 'node:test'
import assert from 'node:assert'

test('sum', (t) => {
  t.test('should add two numbers', () => {
    assert.strictEqual(sum(1, 2), 3)
  })
})

```

Assertions

Assertions are taken care of by the `assert` module, which is built-in

But you can use any other assertion library, like `chai` or `should`

```
import { describe, it } from 'node:test'
import { expect } from 'chai'

describe('sum', () => {
  it('should add two numbers', () => {
    expect(sum(1, 2)).to.equal(3)
  })
})

```

Assertions

Assertions are taken care of by the `assert` module, which is built-in

But you can use any other assertion library, like `chai` or `should`

```
import { describe, it } from 'node:test'
import { expect } from 'chai'

describe('sum', () => {
  it('should add two numbers', () => {
    expect(sum(1, 2)).to.equal(3)
  })
})

```

_the important thing: assertions are not tied to the runner

Running_

Running tests

Node has a built-in `--test` command that you can use to run your tests

```
node --test test.js

```

Running tests

Node has a built-in `--test` command that you can use to run your tests

```
node --test test.js

```

This will give you a TAP output:

```
❯ node --test test.js
  ▶ sum
    ✔ should add two numbers (0.130208ms)
  ▶ sum (0.949ms)

ℹ tests 1
ℹ suites 1
ℹ pass 1
ℹ fail 0
ℹ cancelled 0
ℹ skipped 0
ℹ todo 0
ℹ duration_ms 49.5135

```

Running tests

Node has a built-in `--test` command that you can use to run your tests

```
node --test test.js

```

This will give you a TAP output:

```
❯ node --test test.js
  ▶ sum
    ✔ should add two numbers (0.130208ms)
  ▶ sum (0.949ms)

ℹ tests 1
ℹ suites 1
ℹ pass 1
ℹ fail 0
ℹ cancelled 0
ℹ skipped 0
ℹ todo 0
ℹ duration_ms 49.5135

```

Running tests

Node has a built-in `--test` command that you can use to run your tests

```
node --test test.js

```

This will give you a TAP output:

```
❯ node --test test.js
  ▶ sum
    ✔ should add two numbers (0.130208ms)
  ▶ sum (0.949ms)

ℹ tests 1
ℹ suites 1
ℹ pass 1
ℹ fail 0
ℹ cancelled 0
ℹ skipped 0
ℹ todo 0
ℹ duration_ms 49.5135

```

Running tests

Node has a built-in `--test` command that you can use to run your tests

```
node --test test.js

```

This will give you a TAP output:

```
❯ node --test test.js
  ▶ sum
    ✔ should add two numbers (0.130208ms)
  ▶ sum (0.949ms)

ℹ tests 1
ℹ suites 1
ℹ pass 1
ℹ fail 0
ℹ cancelled 0
ℹ skipped 0
ℹ todo 0
ℹ duration_ms 49.5135

```

Mocks_

Mocking

Today, Node support mocking of some built-in structures:

```
import test from 'node:test'

test('mocks', (ctx) => {
  ctx.mock.fn() // mock function
  ctx.mock.getter() // mock getter
  ctx.mock.setter() // mock setter
  ctx.mock.method() // mock method
})

```

Mocking

Today, Node support mocking of some built-in structures:

```
import test from 'node:test'

test('mocks', (ctx) => {
  ctx.mock.fn() // mock function
  ctx.mock.getter() // mock getter
  ctx.mock.setter() // mock setter
  ctx.mock.method() // mock method
})

```

Mocking

Today, Node support mocking of some built-in structures:

```
import test from 'node:test'

test('mocks', (ctx) => {
  ctx.mock.fn() // mock function
  ctx.mock.getter() // mock getter
  ctx.mock.setter() // mock setter
  ctx.mock.method() // mock method
})

```

Mocking

Today, Node support mocking of some built-in structures:

```
import test from 'node:test'

test('mocks', (ctx) => {
  ctx.mock.fn() // mock function
  ctx.mock.getter() // mock getter
  ctx.mock.setter() // mock setter
  ctx.mock.method() // mock method
})

```

Mocking

Today, Node support mocking of some built-in structures:

```
import test from 'node:test'

test('mocks', (ctx) => {
  ctx.mock.fn() // mock function
  ctx.mock.getter() // mock getter
  ctx.mock.setter() // mock setter
  ctx.mock.method() // mock method
})

```

It also allows you to mock two more complex structures

Mocking

Timers (link)

Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
import assert from 'node:assert'
import { describe, it } from 'node:test'

describe('setTimeout', () => {
  it('should call the callback', (ctx) => {
    const fn = ctx.mock.fn() // look ma, no jest!
    ctx.mock.timers.enable({ apis: ['setTimeout'] })
    setTimeout(fn, 99999)
    assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet

    ctx.mock.timers.tick(99999) // advance the timer
    assert.strictEqual(fn.mock.callCount(), 1) // timer is called

    ctx.mock.timers.reset() // reset back to original state
  })
})

```

Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
1  import assert from 'node:assert'
2  import { describe, it } from 'node:test'
3
4  describe('setTimeout', () => {
5    it('should call the callback', (ctx) => {
6      const fn = ctx.mock.fn() // look ma, no jest!
7      ctx.mock.timers.enable({ apis: ['setTimeout'] })
8      setTimeout(fn, 99999)
9      assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11     ctx.mock.timers.tick(99999) // advance the timer
12     assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14     ctx.mock.timers.reset() // reset back to original state
15   })
16 })
17
```

Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
1  import assert from 'node:assert'
2  import { describe, it } from 'node:test'
3
4  describe('setTimeout', () => {
5    it('should call the callback', (ctx) => {
6      const fn = ctx.mock.fn() // look ma, no jest!
7      ctx.mock.timers.enable({ apis: ['setTimeout'] })
8      setTimeout(fn, 99999)
9      assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11     ctx.mock.timers.tick(99999) // advance the timer
12     assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14     ctx.mock.timers.reset() // reset back to original state
15   })
16 })
17
```

Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
1  import assert from 'node:assert'
2  import { describe, it } from 'node:test'
3
4  describe('setTimeout', () ⇒ {
5    it('should call the callback', (ctx) ⇒ {
6      const fn = ctx.mock.fn() // look ma, no jest!
7      ctx.mock.timers.enable({ apis: ['setTimeout'] })
8      setTimeout(fn, 99999)
9      assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11     ctx.mock.timers.tick(99999) // advance the timer
12     assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14     ctx.mock.timers.reset() // reset back to original state
15   })
16 })
17
```

Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
import assert from 'node:assert'
import { describe, it } from 'node:test'

describe('setTimeout', () => {
  it('should call the callback', (ctx) => {
    const fn = ctx.mock.fn() // look ma, no jest!
    ctx.mock.timers.enable({ apis: ['setTimeout'] })
    setTimeout(fn, 99999)
    assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet

    ctx.mock.timers.tick(99999) // advance the timer
    assert.strictEqual(fn.mock.callCount(), 1) // timer is called

    ctx.mock.timers.reset() // reset back to original state
  })
})

```

Mock Timers

Allows you to fake timers, like `setTimeout` and `setInterval`

```
import assert from 'node:assert'
import { describe, it } from 'node:test'

describe('setTimeout', () => {
  it('should call the callback', (ctx) => {
    const fn = ctx.mock.fn() // look ma, no jest!
    ctx.mock.timers.enable({ apis: ['setTimeout'] })
    setTimeout(fn, 99999)
    assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet

    ctx.mock.timers.tick(99999) // advance the timer
    assert.strictEqual(fn.mock.callCount(), 1) // timer is called

    ctx.mock.timers.reset() // reset back to original state
  })
})

```

Mock Timers

You can also import `mock` directly from `node:test`

```
1  import assert from 'node:assert'
2  import { describe, it, mock } from 'node:test'
3
4  describe('setTimeout', () ⇒ {
5    it('should call the callback', (ctx) ⇒ {
6      const fn = mock.fn() // look ma, no jest!
7      mock.timers.enable({ apis: ['setTimeout'] })
8      setTimeout(fn, 99999)
9      assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet
10
11     mock.timers.tick(99999) // advance the timer
12     assert.strictEqual(fn.mock.callCount(), 1) // timer is called
13
14     mock.timers.reset() // reset back to original state
15   })
16 })
17
```

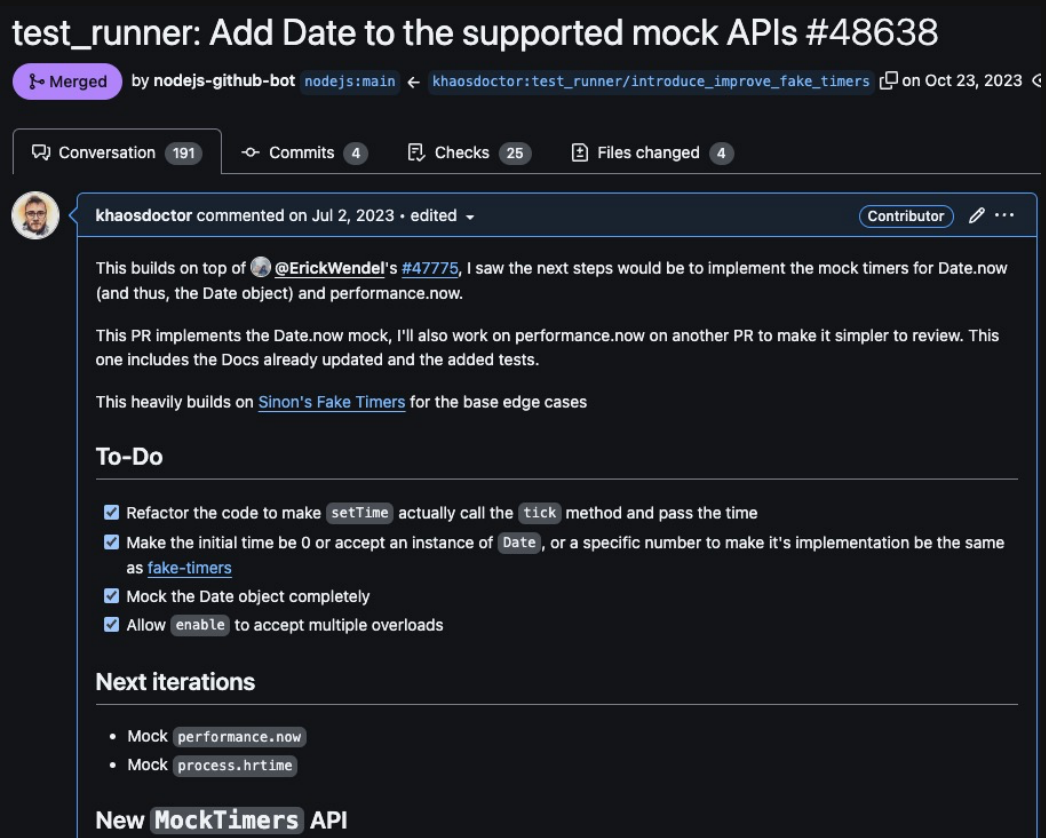
Mock Timers

You can also import `mock` directly from `node:test`

```
import assert from 'node:assert'
import { describe, it, mock } from 'node:test'

describe('setTimeout', () => {
  it('should call the callback', (ctx) => {
    const fn = mock.fn() // look ma, no jest!
    mock.timers.enable({ apis: ['setTimeout'] })
    setTimeout(fn, 99999)
    assert.strictEqual(fn.mock.callCount(), 0) // timer is not called yet

    mock.timers.tick(99999) // advance the timer
    assert.strictEqual(fn.mock.callCount(), 1) // timer is called

    mock.timers.reset() // reset back to original state
  })
})

```

Date mocking

Dates (by yours truly) (link)



test_runner: Add Date to the supported mock APIs #48638

Merged by nodejs-github-bot `nodejs:main` ← `khaosdoctor:test_runner/introduce_improve_fake_timers` on Oct 23, 2023

Conversation 191 · Commits 4 · Checks 25 · Files changed 4

khaosdoctor commented on Jul 2, 2023 • edited — Contributor

This builds on top of @ErickWendel's #47775, I saw the next steps would be to implement the mock timers for Date.now (and thus, the Date object) and performance.now.

This PR implements the Date.now mock, I'll also work on performance.now on another PR to make it simpler to review. This one includes the Docs already updated and the added tests.

This heavily builds on Sinon's Fake Timers for the base edge cases

To-Do

- [x] Refactor the code to make `setTime` actually call the `tick` method and pass the time
- [x] Make the initial time be 0 or accept an instance of `Date`, or a specific number to make it's implementation be the same as fake-timers
- [x] Mock the Date object completely
- [x] Allow `enable` to accept multiple overloads

Next iterations

- Mock `performance.now`
- Mock `process.hrtime`

New `MockTimers` API

Mocking dates

```
import assert from 'node:assert';
import { test } from 'node:test';

test('mocks the Date object', (context) => {
  context.mock.timers.enable({ apis: ['Date'] });
  assert.strictEqual(Date.now(), 0); // Date is mocked to 1970-01-01

  // Advance in time will also advance the date
  context.mock.timers.tick(9999);
  assert.strictEqual(Date.now(), 9999); // Date is mocked to 1970-01-01 + 9999ms
  context.mock.timers.reset() // reset back to original state
});

```

Mocking dates

```
import assert from 'node:assert';
import { test } from 'node:test';

test('mocks the Date object', (context) => {
  context.mock.timers.enable({ apis: ['Date'] });
  assert.strictEqual(Date.now(), 0); // Date is mocked to 1970-01-01

  // Advance in time will also advance the date
  context.mock.timers.tick(9999);
  assert.strictEqual(Date.now(), 9999); // Date is mocked to 1970-01-01 + 9999ms
  context.mock.timers.reset() // reset back to original state
});

```

Mocking dates

```
import assert from 'node:assert';
import { test } from 'node:test';

test('mocks the Date object', (context) => {
  context.mock.timers.enable({ apis: ['Date'] });
  assert.strictEqual(Date.now(), 0); // Date is mocked to 1970-01-01

  // Advance in time will also advance the date
  context.mock.timers.tick(9999);
  assert.strictEqual(Date.now(), 9999); // Date is mocked to 1970-01-01 + 9999ms
  context.mock.timers.reset() // reset back to original state
});

```

Mocking dates

```
import assert from 'node:assert';
import { test } from 'node:test';

test('mocks the Date object', (context) => {
  context.mock.timers.enable({ apis: ['Date'] });
  assert.strictEqual(Date.now(), 0); // Date is mocked to 1970-01-01

  // Advance in time will also advance the date
  context.mock.timers.tick(9999);
  assert.strictEqual(Date.now(), 9999); // Date is mocked to 1970-01-01 + 9999ms
  context.mock.timers.reset() // reset back to original state
});

```

Mocking dates

```
import assert from 'node:assert';
import { test } from 'node:test';

test('mocks the Date object', (context) => {
  context.mock.timers.enable({ apis: ['Date'] });
  assert.strictEqual(Date.now(), 0); // Date is mocked to 1970-01-01

  // Advance in time will also advance the date
  context.mock.timers.tick(9999);
  assert.strictEqual(Date.now(), 9999); // Date is mocked to 1970-01-01 + 9999ms
  context.mock.timers.reset() // reset back to original state
});

```

Code Coverage_

Code coverage

The test runner supports code coverage under an `--experimental-test-coverage` flag

```
node --test --experimental-test-coverage test.js

```

Code coverage

The test runner supports code coverage under an `--experimental-test-coverage` flag

```
node --test --experimental-test-coverage test.js

```

As you can expect, it’s experimental, so use with caution

Code coverage

```
❯ node --test --experimental-test-coverage test.js
▶ sum
  ✔ should add two numbers (0.193833ms)
▶ sum (1.418792ms)

ℹ tests 1
ℹ suites 1
ℹ pass 1
ℹ fail 0
ℹ cancelled 0
ℹ skipped `
ℹ todo 0
ℹ duration_ms 63.954292
ℹ start of coverage report
ℹ ---------------------------------------------------------------
ℹ file      | line % | branch % | funcs % | uncovered lines
ℹ ---------------------------------------------------------------
ℹ index.js |  66.67 |    66.67 |  100.00 | 3-4
ℹ test.js  | 100.00 |   100.00 |  100.00 |
ℹ ---------------------------------------------------------------
ℹ all fil… |  86.67 |    83.33 |  100.00 |
ℹ ---------------------------------------------------------------
ℹ end of coverage report

```

Code coverage

```
❯ node --test --experimental-test-coverage test.js
▶ sum
  ✔ should add two numbers (0.193833ms)
▶ sum (1.418792ms)

ℹ tests 1
ℹ suites 1
ℹ pass 1
ℹ fail 0
ℹ cancelled 0
ℹ skipped `
ℹ todo 0
ℹ duration_ms 63.954292
ℹ start of coverage report
ℹ ---------------------------------------------------------------
ℹ file      | line % | branch % | funcs % | uncovered lines
ℹ ---------------------------------------------------------------
ℹ index.js |  66.67 |    66.67 |  100.00 | 3-4
ℹ test.js  | 100.00 |   100.00 |  100.00 |
ℹ ---------------------------------------------------------------
ℹ all fil… |  86.67 |    83.33 |  100.00 |
ℹ ---------------------------------------------------------------
ℹ end of coverage report

```

Code coverage

```
❯ node --test --experimental-test-coverage test.js
▶ sum
  ✔ should add two numbers (0.193833ms)
▶ sum (1.418792ms)

ℹ tests 1
ℹ suites 1
ℹ pass 1
ℹ fail 0
ℹ cancelled 0
ℹ skipped `
ℹ todo 0
ℹ duration_ms 63.954292
ℹ start of coverage report
ℹ ---------------------------------------------------------------
ℹ file      | line % | branch % | funcs % | uncovered lines
ℹ ---------------------------------------------------------------
ℹ index.js |  66.67 |    66.67 |  100.00 | 3-4
ℹ test.js  | 100.00 |   100.00 |  100.00 |
ℹ ---------------------------------------------------------------
ℹ all fil… |  86.67 |    83.33 |  100.00 |
ℹ ---------------------------------------------------------------
ℹ end of coverage report
```

Code coverage

```
❯ node --test --experimental-test-coverage test.js
▶ sum
  ✔ should add two numbers (0.193833ms)
▶ sum (1.418792ms)

ℹ tests 1
ℹ suites 1
ℹ pass 1
ℹ fail 0
ℹ cancelled 0
ℹ skipped `
ℹ todo 0
ℹ duration_ms 63.954292
ℹ start of coverage report
ℹ ---------------------------------------------------------------
ℹ file      | line % | branch % | funcs % | uncovered lines
ℹ ---------------------------------------------------------------
ℹ index.js |  66.67 |    66.67 |  100.00 | 3-4
ℹ test.js  | 100.00 |   100.00 |  100.00 |
ℹ ---------------------------------------------------------------
ℹ all fil… |  86.67 |    83.33 |  100.00 |
ℹ ---------------------------------------------------------------
ℹ end of coverage report
```

Code coverage

Those uncovered lines are the throw statement in the `sum` function

```
1  export function sum(a, b) {
2    if (typeof a !== 'number' || typeof b !== 'number') {
3      throw new Error('Invalid input')
4    }
5    return a + b
6  }
7
```

Coverage reporters

You can change the coverage reporters with the flag `--test-reporter`

```
node --test --experimental-test-coverage \
--test-reporter=lcov \
--test-reporter-destination=lcov.info \
test.js

```

Coverage reporters

You can change the coverage reporters with the flag `--test-reporter`

```
node --test --experimental-test-coverage \
--test-reporter=lcov \
--test-reporter-destination=lcov.info \
test.js

```

This gets you an `lcov.info` file that you can import into any coverage tool

The test runner supports:

- spec
- dot
- tap
- lcov
- junit

TypeScript 💙

TypeScript support

Node already supports what’s called "importers".

TypeScript support

Node already supports what’s called "importers".

- Importers (previously "loaders") are functions that execute before a module is loaded

TypeScript support

Node already supports what’s called "importers".

- Importers (previously "loaders") are functions that execute before a module is loaded
- They can be used to transform the module before it’s imported

TypeScript support

Node already supports what’s called "importers".

- Importers (previously "loaders") are functions that execute before a module is loaded
- They can be used to transform the module before it’s imported
- This means you can transpile TypeScript on the fly

TypeScript support

Node already supports what's called "importers".

- Importers (previously "loaders") are functions that execute before a module is loaded
- They can be used to transform the module before it's imported
- This means you can transpile TypeScript on the fly
- Common importers are `ts-node`, `tsx`, and `esbuild`

TypeScript support

Let's take this example:

```ts
// index.ts
export function sum(a: number, b: number) {
  if (typeof a !== 'number' || typeof b !== 'number') {
    throw new Error('Invalid input')
  }
  return a + b
}

console.log(sum(1, 2))

```

TypeScript support

Let's take this example:

```ts
// index.ts
export function sum(a: number, b: number) {
  if (typeof a !== 'number' || typeof b !== 'number') {
    throw new Error('Invalid input')
  }
  return a + b
}

console.log(sum(1, 2))

```

Install `tsx` with `npm install tsx` and run the file with:

```
node --import=tsx index.ts

```

TypeScript support

You will get this

```
❯ node --import=tsx index.ts
3

```

TypeScript support

You will get this

```
❯ node --import=tsx index.ts
3

```

WHAT IF I TOLD YOU…

TypeScript support

You will get this

```
❯ node --import=tsx index.ts
3

```

WHAT IF I TOLD YOU…

you can use this with the test runner?

TypeScript support

Let’s just take the previous example and change the file extension to `test.ts`

Then run it with the `--import` flag

TypeScript support

Let's just take the previous example and change the file extension to `test.ts`

Then run it with the `--import` flag

```
❯ node --import=tsx --test test.ts
3 # our console
▶ sum
  ✔ should add two numbers (0.128ms)
▶ sum (0.72875ms)

ℹ tests 1
ℹ suites 1
ℹ pass 1
ℹ fail 0
ℹ cancelled 0
ℹ skipped 0
ℹ todo 0
ℹ duration_ms 157.107959

```

TypeScript support

Let's just take the previous example and change the file extension to `test.ts`

Then run it with the `--import` flag

```
❯ node --import=tsx --test test.ts
3 # our console
▶ sum
  ✔ should add two numbers (0.128ms)
▶ sum (0.72875ms)

ℹ tests 1
ℹ suites 1
ℹ pass 1
ℹ fail 0
ℹ cancelled 0
ℹ skipped 0
ℹ todo 0
ℹ duration_ms 157.107959

```

Now look at that `jest.config.js` file you've been maintaining as a boilerplate for 5 years and tell me if this isn't just a bliss

Other supported features

Other supported features

- Aborting tests via abort signals

Other supported features

- Aborting tests via abort signals
- Skipping tests with `t/it.skip`

Other supported features

- Aborting tests via abort signals
- Skipping tests with `t/it.skip`
- Todo tests with `t/it.todo`

Other supported features

- Aborting tests via abort signals
- Skipping tests with `t/it.skip`
- Todo tests with `t/it.todo`
- "only" flag to run only a specific test with `t/it.only`

Other supported features

- Aborting tests via abort signals
- Skipping tests with `t/it.skip`
- Todo tests with `t/it.todo`
- "only" flag to run only a specific test with `t/it.only`
- Lifecycle hooks like `before`, `after`, `beforeEach`, `afterEach`

Other supported features

- Aborting tests via abort signals
- Skipping tests with `t/it.skip`
- Todo tests with `t/it.todo`
- "only" flag to run only a specific test with `t/it.only`
- Lifecycle hooks like `before`, `after`, `beforeEach`, `afterEach`
- Test filtering with `--test-name-pattern` via RegExp

Other supported features

- Aborting tests via abort signals
- Skipping tests with `t/it.skip`
- Todo tests with `t/it.todo`
- "only" flag to run only a specific test with `t/it.only`
- Lifecycle hooks like `before`, `after`, `beforeEach`, `afterEach`
- Test filtering with `--test-name-pattern` via RegExp
- Watch mode (experimental)

The future of Node.js Testing

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

← Home

A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

Improve Filtering

The future of Node.js Testing

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon

A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

Improve Filtering

The future of Node.js Testing

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering

A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

Improve Filtering

The future of Node.js Testing

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering
- (maybe) More reporters

A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

Improve Filtering

The future of Node.js Testing

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering
- (maybe) More reporters
- Snapshot testing

A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

Improve Filtering

The future of Node.js Testing

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering
- (maybe) More reporters
- Snapshot testing
- Source map support

A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

Improve Filtering

The future of Node.js Testing

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering
- (maybe) More reporters
- Snapshot testing
- Source map support
- MOAR mocks! (like `performance`)

A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

Improve Filtering

The future of Node.js Testing

Funny enough, Colin Ihrig (one of the main contributors to the test runner) has a wishlist for 2024.

- Module mocking is something we might see soon
- Improved filtering
- (maybe) More reporters
- Snapshot testing
- Source map support
- MOAR mocks! (like `performance`)
- The **definitive** downfall of Jest (just kidding)

A 2024 Wishlist for Node's Test Runner

Colin J. Ihrig
2024-01-09

When the Node.js test runner was initially created, it was supposed to be very minimal. Of course, developers are never happy with simple and/or minimal, so the test runner has already grown far beyond what it was ever intended to be. Since that trend seems likely to continue, I have put together a short wishlist of improvements that I would love to see in Node's test runner even though I have moved on from Node.js myself.

Improve Filtering

Get involved

The test runner is still in active development, and you can get involved!

Get involved

The test runner is still in active development, and you can get involved!

- Documentation is always a good place to start

Get involved

The test runner is still in active development, and you can get involved!

- Documentation is always a good place to start
- You can also contribute to the code

Get involved

The test runner is still in active development, and you can get involved!

- Documentation is always a good place to start
- You can also contribute to the code
  - Maybe making Colin happy with his wishlist

Get involved

The test runner is still in active development, and you can get involved!

- Documentation is always a good place to start
- You can also contribute to the code
  - Maybe making Colin happy with his wishlist
- Or just use it and give feedback!

Get involved

The test runner is still in active development, and you can get involved!

- Documentation is always a good place to start
- You can also contribute to the code
  - Maybe making Colin happy with his wishlist
- Or just use it and give feedback!
- Start shifting your projects to it, and help the community grow!

Thank you!

https://{twitter,instagram,github,youtube,linkedin}.lsantos.dev

See this talk on my website

https://lsantos.dev/talks/nodejs-test-runner