Web Platform Tests

The tests here are drivers for running the Web Platform Tests.

See `test/fixtures/wpt/README.md` for a hash of the last updated WPT commit for each module being covered here.

See the json files in the `status` folder for prerequisites, expected failures, and support status for specific tests in each module.

Currently there are still some Web Platform Tests titled `test-whatwg-*` under `test/parallel` that have not been migrated to be run with the WPT harness and have automatic updates. There are also a few `test-whatwg-*-custom-*` tests that may need to be upstreamed. This folder covers the tests that have been migrated.

How to add tests for a new module

1. Create a status file

For example, to add the URL tests, add a `test/wpt/status/url.json` file.

In the beginning, it's fine to leave an empty object `{}` in the file if it's not yet clear how compliant the implementation is, the requirements and expected failures can be figured out in a later step when the tests are run for the first time.

See Format of a status JSON file for details.

2. Pull the WPT files

Use the git node wpt command to download the WPT files into `test/fixtures/wpt`. For example, to add URL tests:

```
$ cd /path/to/node/project
$ git node wpt url
```

3. Create the test driver

For example, for the URL tests, add a file `test/wpt/test-url.js`:

```
'use strict';

require('../common');
const { WPTRunner } = require('../common/wpt');

const runner = new WPTRunner('url');

// Set Node.js flags required for the tests.
runner.setFlags(['--expose-internals']);
```

```
// Set a script that will be executed in the worker before running the tests.
runner.setInitScript(`
  const { internalBinding } = require('internal/test/binding');
  const { DOMException } = internalBinding('messaging');
  global.DOMException = DOMException;
`);

runner.runJsTests();
```

This driver is capable of running the tests located in `test/fixtures/wpt/url` with the WPT harness while taking the status file into account.

4. Run the tests

Run the test using `tools/test.py` and see if there are any failures. For example, to run all the URL tests under `test/fixtures/wpt/url`:

```
$ tools/test.py wpt/test-url
```

To run a specific test in WPT, for example, `url/url-searchparams.any.js`, pass the file name as argument to the corresponding test driver:

```
node test/wpt/test-url.js url-searchparams.any.js
```

If there are any failures, update the corresponding status file (in this case, `test/wpt/status/url.json`) to make the test pass.

For example, to mark `url/url-searchparams.any.js` as expected to fail, add this to `test/wpt/status/url.json`:

```
  "url-searchparams.any.js": {
    "fail": "explain why the test fails, ideally with links"
  }
```

See Format of a status JSON file for details.

5. Commit the changes and submit a Pull Request

See the contributing guide.

How to update tests for a module

The tests can be updated in a way similar to how they are added. Run Step 2 and Step 4 of adding tests for a new module.

The git node wpt command maintains the status of the local WPT subset, if no files are updated after running it for a module, the local subset is up to date and there is no need to update them until they are changed in the upstream.

How it works

Note: currently this test suite only supports `.js` tests. There is ongoing work in the upstream to properly split out the tests into files that can be run in a shell environment like Node.js.

Getting the original test files and harness from WPT

The original files and harness from WPT are downloaded and stored in `test/fixtures/wpt`.

The git node wpt command automate this process while maintaining a map containing the hash of the last updated commit for each module in `test/fixtures/wpt/versions.json` and `test/fixtures/wpt/README.md`. It also maintains the LICENSE file in `test/fixtures/wpt`.

Loading and running the tests

Given a module, the `WPTRunner` class in `test/common/wpt` loads:

- `.js` test files (for example, `test/common/wpt/url/*.js` for `url`)
- Status file (for example, `test/wpt/status/url.json` for `url`)
- The WPT harness

Then, for each test, it creates a worker thread with the globals and mocks, sets up the harness result hooks, loads the metadata in the test (including loading extra resources), and runs all the tests in that worker thread, skipping tests that cannot be run because of lack of dependency or expected failures.

Format of a status JSON file

```
{
  "something.scope.js": {  // the file name
    // Optional: If the requirement is not met, this test will be skipped
    "requires": ["small-icu"],  // supports: "small-icu", "full-icu"

    // Optional: the test will be skipped with the reason printed
    "skip": "explain why we cannot run a test that's supposed to pass",

    // Optional: the test will be skipped with the reason printed
    "fail": "explain why we the test is expected to fail"
  }
}
```

A test may have to be skipped because it depends on another irrelevant Web API, or certain harness has not been ported in our test runner yet. In that case it needs to be marked with `skip` instead of `fail`.