Parallelism

**Introduction**

In Playwright, tests are typically run in parallel to optimize performance and reduce execution time. By default, Playwright Test executes multiple test files in parallel using separate worker processes. Each worker runs independently, providing an isolated environment to avoid conflicts between tests. However, Playwright also offers the flexibility to run tests in serial mode when test dependencies exist.

This chapter explores how to configure and manage parallel and serial test execution in Playwright.

**Parallel Execution**

Parallel execution in Playwright allows test files to be run simultaneously in different worker processes. These workers are independent OS processes, and each worker starts its own browser instance. Parallelism is enabled by default, but you can adjust the level of parallelism based on your project needs.

**Configuring Parallel Tests**

1. **Default Behavior**: Test files are run in parallel, while individual tests inside a file run sequentially.
2. **Configuring Tests in a Single File**: You can configure tests inside a single file to run in parallel by using test.describe.configure({ mode: 'parallel' }).

typescript

Copy code

import { test } from '@playwright/test';

test.describe.configure({ mode: 'parallel' });

test('runs in parallel 1', async ({ page }) => {

// Test logic

});

test('runs in parallel 2', async ({ page }) => {

// Test logic

});

1. **Configuring Entire Projects**: You can configure an entire project to run all tests in parallel by setting fullyParallel in the Playwright configuration file:

typescript

Copy code

import { defineConfig } from '@playwright/test';

export default defineConfig({

fullyParallel: true, // Enable parallel execution for all tests

});

**Limiting Parallel Workers**

You can control the maximum number of parallel worker processes to run tests more efficiently, especially in CI environments or resource-limited systems. This can be done either via command line or in the configuration file.

* **Command Line**:

bash

Copy code

npx playwright test --workers=4

* **Configuration File**:

typescript

Copy code

import { defineConfig } from '@playwright/test';

export default defineConfig({

workers: process.env.CI ? 2 : undefined, // Limits workers in CI

});

**Disabling Parallelism**

To disable parallel execution, set the number of workers to 1:

* **Command Line**:

bash

Copy code

npx playwright test --workers=1

* **Configuration File**:

typescript

Copy code

import { defineConfig } from '@playwright/test';

export default defineConfig({

workers: 1, // Disable parallel execution

});

**Serial Execution**

Serial execution is useful for interdependent tests that cannot run in parallel. In serial mode, tests are run one after the other, and if any test fails, the subsequent tests are skipped. This ensures that failures in earlier tests don't compromise the execution of related tests.

**Configuring Serial Tests**

To run tests serially, use test.describe.configure({ mode: 'serial' }). This configuration ensures that tests in the describe block are executed one by one.

Example of serial execution:

typescript

Copy code

import { test } from '@playwright/test';

test.describe.configure({ mode: 'serial' });

test('runs first', async ({ page }) => {

await page.goto('https://playwright.dev/');

});

test('runs second', async () => {

// Interdependent test logic

});

**Why Avoid Serial Tests?**

Serial tests are generally discouraged because they tend to increase the execution time and make test suites harder to maintain. The preferred approach is to design independent tests that can be executed in parallel. However, serial tests might be necessary for certain workflows where order or state dependencies are crucial.

**Sharding Tests Across Machines**

Parallel execution can be further optimized by **sharding** your test suite, splitting it into multiple chunks and running them on different machines.

* **Command Line Example**:

bash

Copy code

npx playwright test --shard=1/3

This splits the test suite into three parts and runs only the first part on the current machine.

**Summary**

Parallel execution is the default and recommended approach in Playwright as it speeds up testing by utilizing multiple worker processes. Serial execution should be used sparingly, only when tests depend on the outcome or state of previous tests. Proper configuration of workers, test modes, and sharding can significantly improve testing efficiency and reliability in larger test suites.

Does this structure align with your expectations? Let me know if you'd like any adjustments or further details!