集成到 Playwright

复制 Markdown 打开

Playwright.js↗ 是由微软开发的一个开源自动化库，主要用于对网络应用程序进行端到端测试（end-to-end test）和网页抓取。

与 Playwright 的集成方式有以下两种方式：

- 直接用脚本方式集成和调用 Midscene Agent，适合快速体验、原型开发、数据抓取和自动化脚本等场景。
- 在 Playwright 的测试用例中集成 Midscene，适合需要执行 UI 测试的场景。

配置 AI 模型服务

将你的模型配置写入环境变量，可参考 **模型策略** 了解更多细节。

```
export MIDSCENE_MODEL_BASE_URL="https://替换为你的模型服务地址/v1"
export MIDSCENE_MODEL_API_KEY="替换为你的 API Key"
export MIDSCENE_MODEL_NAME="替换为你的模型名称"
export MIDSCENE_MODEL_FAMILY="替换为你的模型系列"
```

更多配置信息请参考 **模型策略** 和 **模型配置**。

直接集成 Midscene Agent

样例项目

你可以在这里看到向 Playwright 集成的样例项目：https://github.com/web-infra-dev/midscene-example/blob/main/playwright-demo ↗

第一步：安装依赖

npm | yarn | pnpm | bun | deno

```
npm install @midscene/web playwright @playwright/test tsx --save-dev
```

第二步：编写脚本

编写下方代码，保存为 `./demo.ts`

```
import { chromium } from 'playwright';
import { PlaywrightAgent } from '@midscene/web/playwright';
import 'dotenv/config'; // read environment variables from .env file

const sleep = (ms) => new Promise((r) => setTimeout(r, ms));

Promise.resolve(
  (async () => {
    const browser = await chromium.launch({
      headless: true, // 'true' means we can't see the browser window
      args: ['--no-sandbox', '--disable-setuid-sandbox'],
    });

    const page = await browser.newPage();
    await page.setViewportSize({
      width: 1280,
      height: 768,
    });
    await page.goto('https://www.ebay.com');
    await sleep(5000); // 👀 init Midscene agent
    const agent = new PlaywrightAgent(page);

    // 👀 type keywords, perform a search
    await agent.aiAct('type "Headphones" in search box, hit Enter');

    // 👀 wait for the loading
    await agent.aiWaitFor('there is at least one headphone item on page');
    // or you may use a plain sleep:
    // await sleep(5000);

    // 👀 understand the page content, find the items
    const items = await agent.aiQuery(
      '{itemTitle: string, price: Number}[], find item in list and corresponding pric
    );
    console.log('headphones in stock', items);

    const isMoreThan1000 = await agent.aiBoolean(
      'Is the price of the headphones more than 1000?',
    );
    console.log('isMoreThan1000', isMoreThan1000);

    const price = await agent.aiNumber(
      'What is the price of the first headphone?',
```

```
    );
    console.log('price', price);

    const name = await agent.aiString(
      'What is the name of the first headphone?',
    );
    console.log('name', name);

    const location = await agent.aiLocate(
      'What is the location of the first headphone?',
    );
    console.log('location', location);

    // 👀 assert by AI
    await agent.aiAssert('There is a category filter on the left');

    // 👀 click on the first item
    await agent.aiTap('the first item in the list');

    await browser.close();
  })(),
);
```

更多 Agent 的 API 讲解请参考 **API 参考**。

第三步：运行

使用 `tsx` 来运行，你会看到命令行打印出了耳机的商品信息：

```
# run
npx tsx demo.ts

# 命令行应该有如下输出
#  [
#    {
#      itemTitle: 'JBL Tour Pro 2 - True wireless Noise Cancelling earbuds with Smart
#      price: 551.21
#    },
#    {
#      itemTitle: 'Soundcore Space One无线耳机40H ANC播放时间2XStronger语音还原',
#      price: 543.94
#    }
# ]
```

第四步：查看运行报告

当上面的命令执行成功后，会在控制台输出： `Midscene - report file updated: /path/to/report/some_id.html` ，通过浏览器打开该文件即可看到报告。

在 Playwright 的测试用例中集成 Midscene

这里我们假设你已经拥有一个集成了 Playwright 的测试项目。

样例项目

你可以在这里看到向 Playwright 集成的样例项目：https://github.com/web-infra-dev/midscene-example/blob/main/playwright-testing-demo↗

第一步：新增依赖，更新配置文件

新增依赖

npm yarn pnpm bun deno

```
npm install @midscene/web --save-dev
```

更新 playwright.config.ts

```
export default defineConfig({
  testDir: './e2e',
+ timeout: 90 * 1000,
+ reporter: [["list"], ["@midscene/web/playwright-reporter", { type: "merged" }]], //
});
```

其中 `reporter` 配置项的 `type` 可选值为 `merged` 或 `separate` ，默认值为 `merged` ，表示多个测试用例生成一个报告，可选值为 `separate` ，表示为每个测试用例一个报告。

第二步：扩展 test 实例

把下方代码保存为 `./e2e/fixture.ts` ；

```
import { test as base } from '@playwright/test';
import type { PlayWrightAiFixtureType } from '@midscene/web/playwright';
import { PlaywrightAiFixture } from '@midscene/web/playwright';

export const test = base.extend<PlayWrightAiFixtureType>(
  PlaywrightAiFixture({
    waitForNetworkIdleTimeout: 2000, // 可选，交互过程中等待网络空闲的超时时间，默认值为
  }),
);
```

第三步：编写测试用例

完整的交互、查询和辅助 API 请参考 **Agent API 参考**。如果需要调用更底层的能力，可以使用 `agentForPage` 获取 `PageAgent` 实例，再直接调用对应的方法：

```
test('case demo', async ({ agentForPage, page }) => {
  const agent = await agentForPage(page);

  await agent.recordToReport();
  const logContent = agent._unstableLogContent();
  console.log(logContent);
});
```

示例代码

./e2e/ebay-search.spec.ts

```
import { expect } from '@playwright/test';
import { test } from './fixture';

test.beforeEach(async ({ page }) => {
  page.setViewportSize({ width: 400, height: 905 });
  await page.goto('https://www.ebay.com');
  await page.waitForLoadState('networkidle');
});

test('search headphone on ebay', async ({
  ai,
  aiQuery,
  aiAssert,
  aiInput,
  aiTap,
  aiScroll,
  aiWaitFor,
  aiRightClick,
  recordToReport,
}) => {
  // 使用 aiInput 输入搜索关键词
  await aiInput('Headphones', '搜索框');

  // 使用 aiTap 点击搜索按钮
  await aiTap('搜索按钮');

  // 等待搜索结果加载
  await aiWaitFor('搜索结果列表已加载', { timeoutMs: 5000 });

  // 使用 aiScroll 滚动到页面底部
  await aiScroll(
    {
      scrollType: 'untilBottom',
    },
    '搜索结果列表',
  );

  // 使用 aiQuery 获取商品信息
  const items =
    await aiQuery<Array<{ title: string; price: number }>>(
      '获取搜索结果中的商品标题和价格',
    );
```

```
  console.log('headphones in stock', items);
  expect(items?.length).toBeGreaterThan(0);

  // 使用 aiAssert 验证筛选功能
  await aiAssert('界面左侧有类目筛选功能');

  // 使用 recordToReport 记录当前状态
  await recordToReport('搜索结果', { content: '耳机搜索的最终结果' });
});
```

更多 Agent 的 API 讲解请参考 **API 参考**。

第四步：运行测试用例

```
npx playwright test ./e2e/ebay-search.spec.ts
```

第五步：查看测试报告

当上面的命令执行成功后，会在控制台输出： `Midscene - report file updated: ./current_cwd/midscene_run/report/some_id.html` ，通过浏览器打开该文件即可看到报告。

Advanced

关于在新标签页打开

每个 Agent 实例都与对应的页面唯一绑定，为了方便开发者调试，Midscene 默认拦截了新 tab 的页面（如点击一个带有 `target="_blank"` 属性的链接），将其改为在当前页面打开。

如果你想恢复在新标签页打开的行为，你可以设置 `forceSameTabNavigation` 选项为 `false` ，但相应的，你需要为新标签页创建一个 Agent 实例。

```
const mid = new PlaywrightAgent(page, {
  forceSameTabNavigation: false,
});
```

连接远程 Playwright 浏览器并接入 Midscene Agent

示例项目

你可以在这里找到远程 Playwright 集成的示例项目：https://github.com/web-infra-dev/midscene-example/tree/main/remote-playwright-demo ↗

当你已经在自有基础设施或供应商服务里运行浏览器时，可通过连接远程 Playwright 服务复用这些浏览器，让实例更贴近目标环境、避免重复启动，同时保持相同的 Midscene AI 自动化能力。

前置依赖

npm | yarn | pnpm | bun | deno

```
npm install playwright @playwright/test @midscene/web --save-dev
```

获取 CDP WebSocket URL

你可以从多种来源获取 CDP WebSocket URL：

- BrowserBase：在 https://browserbase.com ↗ 注册并获取你的 CDP URL
- Browserless：使用 https://browserless.io ↗ 或运行你自己的实例
- **本地 Chrome**：使用 `--remote-debugging-port=9222` 参数运行 Chrome，然后使用 `ws://localhost:9222/devtools/browser/...`
- **Docker**：在 Docker 容器中运行 Chrome 并暴露调试端口

代码示例

```
import { chromium } from 'playwright';
import { PlaywrightAgent } from '@midscene/web/playwright';

// 来自远程浏览器服务的 CDP WebSocket URL
const cdpWsUrl = 'ws://your-remote-browser.com/devtools/browser/your-session-id';

// 连接并选取页面
const browser = await chromium.connectOverCDP(cdpWsUrl);
const context = browser.contexts()[0];
const page = context.pages()[0] || await context.newPage();

// 创建 Midscene Agent（用法与本地 Playwright agent 一致）
const agent = new PlaywrightAgent(page);

// 像平常一样调用 AI 方法
await agent.aiAction('跳转到 https://example.com');
await agent.aiAction('点击登录按钮');

// 清理
await agent.destroy();
await browser.close();
```

连接完成后，后续的 `PlaywrightAgent` 使用方式与本地启动的浏览器保持一致。

扩展自定义交互动作

使用 `customActions` 选项，结合 `defineAction` 定义的自定义交互动作，可以扩展 Agent 的动作空间。这些动作会追加在内置动作之后，方便 Agent 在规划阶段调用。

```ts
import { getMidsceneLocationSchema, z } from '@midscene/core';
import { defineAction } from '@midscene/core/device';

const ContinuousClick = defineAction({
  name: 'continuousClick',
  description: 'Click the same target repeatedly',
  paramSchema: z.object({
    locate: getMidsceneLocationSchema(),
    count: z
      .number()
      .int()
      .positive()
      .describe('How many times to click'),
  }),
  async call(param) {
    const { locate, count } = param;
    console.log('click target center', locate.center);
    console.log('click count', count);
    // 在这里结合 locate + count 实现自定义点击逻辑
  },
});

const agent = new PlaywrightAgent(page, {
  customActions: [ContinuousClick],
});

await agent.aiAct('点击红色按钮五次');
```

更多关于自定义动作的细节，请参考 **集成到任意界面**。

更多

- 更多 Agent 的 API 文档请参考 **API 参考**。
- Playwright 的 API 文档请参考 **Playwright Agent API**↗。
- 样例项目：**直接集成 Playwright**↗，**Playwright 测试集成**↗，**远程 Playwright 集成**↗

上一页
通过 Chrome 插件快速体验

下一页
集成到 Puppeteer