Handling Frames in Playwright

What Are Frames?

Frames are HTML elements and it's a part of a web page that display another web page inside them. They help organize content or show information from different sources.

Identifying Frames

To identify whether an element is a frame or not, follow these steps:

Find Total Number of Frames on a Page

- Use the following command to get an array of all frames on the page:

```
const allFrames = await page.frames();
console.log('Total number of frames:', allFrames.length);
```

- This will print the total number of frames. For example, if there are 8 frames, it will show:

```
Total number of frames: 8
```

Interacting with Frames

There are two main ways to interact with elements inside frames:

1. **Using Frame Object**
2. **Using frameLocator**

Within frame object, you can interact in two ways:

1.By using the **name** of the frame.

2.By using the **URL** of the frame.

Note: If the frame tag contains a **name** or **URL**, it is recommended to use the frame object method. Example:

Example DOM:

```
<frame name="frameName" src="https://advertismentFrameOne.com">
<input type="text" class="userClassFromFrame">
</frame>
```

Approach 1: Using Frame Object

Using Frame Name

```
const { test, expect } = require('@playwright/test');

test('Fill input inside a frame', async ({ page }) => {
    await page.goto('https://your-main-page.com');
    const frameOne = await page.frame({ name: 'frameName' });
    await frameOne.locator('.userClassFromFrame').fill('Yogesh');
    await
expect(frameOne.locator('.userClassFromFrame')).toHaveValue('Yogesh');
});
```

Using Frame URL

```
const { test, expect } = require('@playwright/test');

test('Fill input inside a frame using src URL', async ({ page }) => {
    await page.goto('https://your-main-page.com');
    const frameOne = page.frame({ url: 'https:// advertismentFrameOne.com'
});
    await frameOne.locator('.userClassFromFrame').fill('Yogesh');
    await
expect(frameOne.locator('.userClassFromFrame')).toHaveValue('Yogesh');
});
```

Approach 2: Using frameLocator()

```
const { test, expect } = require('@playwright/test');

test('Fill input inside a frame using frameLocator()', async ({ page }) =>
{
    await page.goto('https://your-main-page.com');
    const frame = page.frameLocator('frame[src="https://
advertismentFrameOne.com"]');
    await frame.locator('.userClassFromFrame').fill('Yogesh');
    await
expect(frame.locator('.userClassFromFrame')).toHaveValue('Yogesh');
});
```

Note: In Playwright, you don't need to explicitly exit the frame after performing an action. You can simply continue interacting with the main page by referencing **page** instead of the frame object.

Handling Nested (Inner) Frames

In some cases, a parent frame may contain multiple child frames (nested frames). To interact with elements inside a specific child frame, follow these steps:

1. Switch to the **parent frame**.
2. Use **childFrames()** and an index to access the desired child frame.
3. Locate and interact with elements inside that child frame.

Example DOM Structure

```
<frame name="parentFrame" src="parent.html">
    <frame name="childFrame1" src="child1.html"></frame>
    <frame name="childFrame2" src="child2.html">
        <input type="radio" name="option" id="radio1">
        <input type="radio" name="option" id="radio2">
    </frame>
    <frame name="childFrame3" src="child3.html"></frame>
</frame>
```

Handling Nested Frames with ChildFrames()

```
const { test, expect } = require('@playwright/test');

test('Handle nested frames using childFrames()', async ({ page }) => {
    await page.goto('https://your-main-page.com');
    const parentFrame = await page.frame({ name: 'parentFrame' });
    const secondChildFrame = parentFrame.childFrames()[1];
    await secondChildFrame.locator('#radio1').click();
    await expect(secondChildFrame.locator('#radio1')).toBeChecked();
});
```

Summary

- Identify frames by checking the total number of frames on the page.
- Use **frameLocator** or **frame object** to interact with elements inside frames.
- Handle nested frames by switching to the parent frame first and then accessing child frames.
- No need to explicitly come out of a frame after interaction.