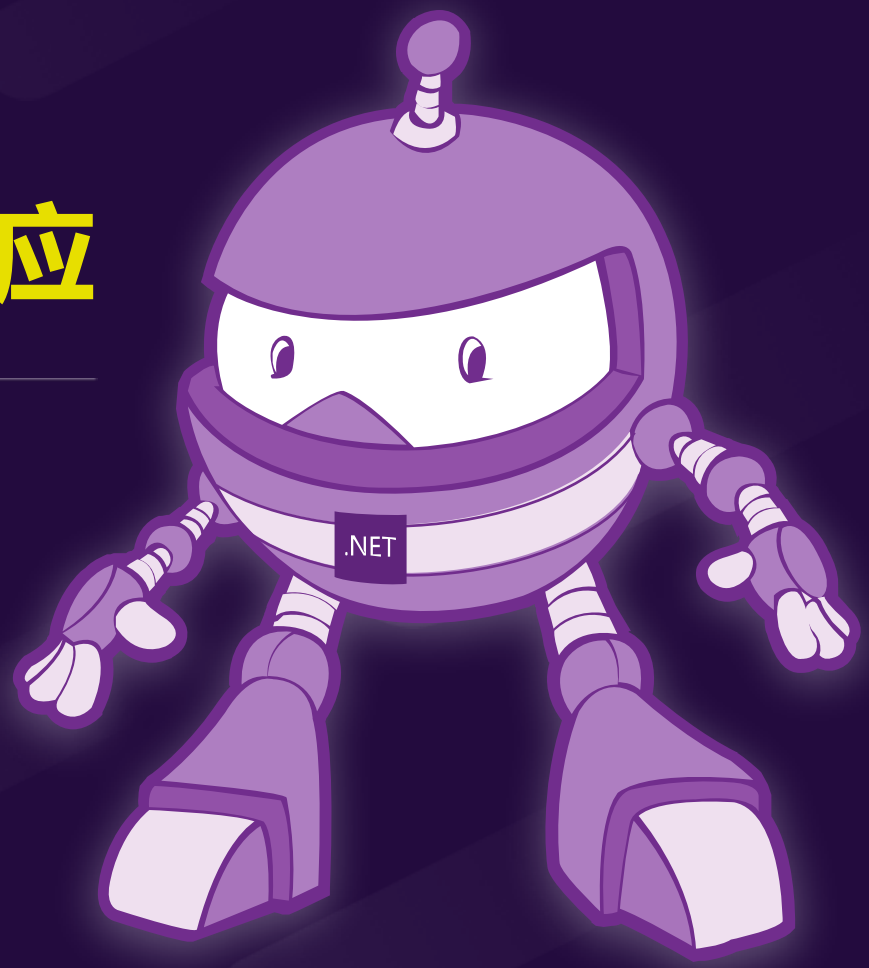
2019中国.NET开发者峰会
China .Net Conf 2019

.NET Core 2019

Puppeteer在.NET中的应用与避坑

演讲人：衣明志



Puppeteer是什么?

Puppeteer 是一个 Google 开源的 NodeJS 库

它提供了一个高级 API 来通过 DevTools 协议控制 Chromium (或 Chrome)

Puppeteer 默认以无头 (Headless) 模式运行，但是可以通过修改配置运行 有头"模式。

能做什么?

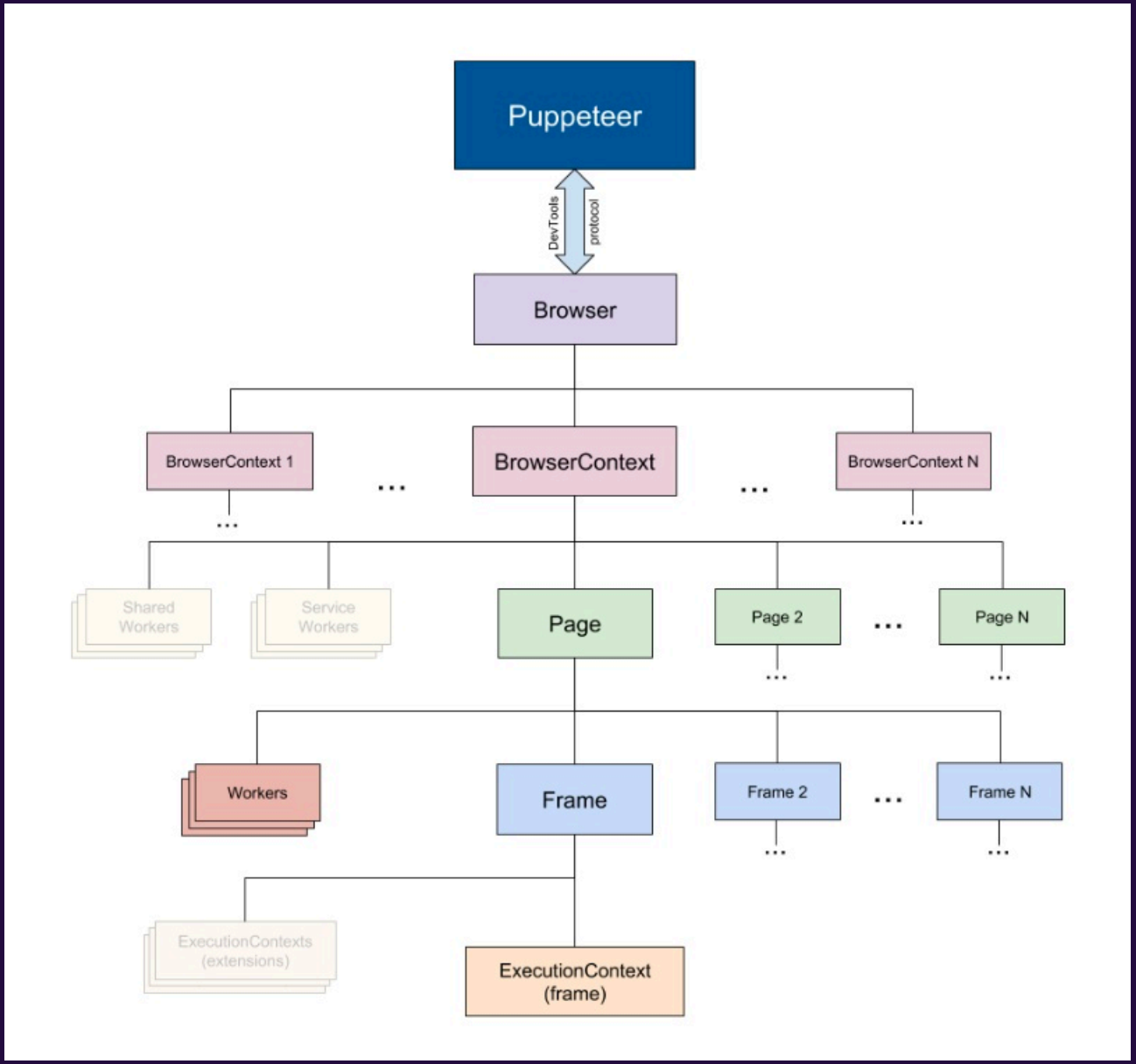
在浏览器中手动可执行的绝大多数操作都可以使用 Puppeteer 来完成!

- 生成页面 PDF 或 图片
- 抓取 SPA（单页应用）并生成预渲染内容，即 “SSR”（服务器端渲染）
- 自动提交表单，进行 UI 测试，键盘输入等
- 创建一个时时更新的自动化测试环境，使用最新的 JavaScript 和浏览器功能直接在最新版本的Chrome中执行测试
- 捕获网站的 timeline trace，用来帮助分析性能问题
- 测试浏览器扩展

做爬虫可好?

技术上可以，但是要注意不要违法。

Puppeteer 结构图



Puppeteer 结构图

- **Puppeteer**
  使用 Dev Tools 协议与浏览器进行通信
- **Browser**
  实例可以拥有浏览器上下文
- **Browser Context**
  实例定义了一个浏览会话并可拥有多个页面
- **Page**
  至少有一个主框架(Main Frame), 可能还有其他框架由 iframe 或 frame 创建
- **Frame**
  至少有一个执行上下文(默认的执行JavaScript的上下文), 框架可能有额外的与扩展关联的执行上下文
- **Worker**
  具有单一执行上下文，以便于和 Web Workers 交互

在.NET下的使用

PuppeteerSharp 2.0.0

Headless Chrome .NET API

Package Manager | .NET CLI | PackageReference | Paket CLI

```
> dotnet add package PuppeteerSharp --version 2.0.0
```

```
await new BrowserFetcher().DownloadAsync(BrowserFetcher.DefaultRevision);  下载浏览器执行程序
var browser = await Puppeteer.LaunchAsync(new LaunchOptions  创建一个浏览器实例
{
    Headless = true   以无头方式运行
});
var page = await browser.NewPageAsync();   打开一个页面
await page.GoToAsync("http://www.google.com");  打开一个网站
await page.ScreenshotAsync(outputFile);   将网站截屏
```

设置浏览区域尺寸


```
await page.SetViewportAsync(new ViewPortOptions
{
    Width = 500,
    Height = 500
});
```

导出PDF


```
await new BrowserFetcher().DownloadAsync(BrowserFetcher.DefaultRevision);
var browser = await Puppeteer.LaunchAsync(new LaunchOptions
{
    Headless = true
});
var page = await browser.NewPageAsync();
await page.GoToAsync("http://www.google.com");
await page.PdfAsync(outputFile);
```

页面注入HTML


```
using(var page = await browser.NewPageAsync())
{
    await page.SetContentAsync("<div>My Receipt</div>");
    var result = await page.GetContentAsync();
    await page.PdfAsync(outputFile);
    SaveHtmlToDB(result);
}
```

页面执行js


```
using (var page = await browser.NewPageAsync())
{
    var seven = await page.EvaluateExpressionAsync<int>("4 + 3");
    var someObject = await page.EvaluateFunctionAsync<dynamic>("(value) => ({a: value})", 5);
    Console.WriteLine(someObject.a);
}
```

链接远程浏览器


```
var options = new ConnectOptions()
{
    BrowserWSEndpoint = $"wss://www.externalbrowser.io?token={apikey}"
};

var url = "https://www.google.com/";

using (var browser = await PuppeteerSharp.Puppeteer.ConnectAsync(options))
{
    using (var page = await browser.NewPageAsync())
    {
        await page.GoToAsync(url);
        await page.PdfAsync("wot.pdf");
    }
}
```

坑: Docker中执行


```
_browser = await Puppeteer.LaunchAsync(new LaunchOptions
{
    Headless = true,
    Args = new[]{"--no-sandbox","--disable-setuid-sandbox"}
});
```

坑: 缓存影响


```
await page.SetCacheEnabledAsync(false).ConfigureAwait(false);
```

坑: SPA &Ajax


```
await page.GoToAsync(url, 30000, new[] { WaitUntilNavigation.Networkidle0 }).ConfigureAwait(false);
```

`waitUntil` <string|Array<string>> 满足什么条件认为页面跳转完成，默认是 `load` 事件触发时。指定事件数组，那么所有事件触发后才认为是跳转完成。事件包括：

- `load` - 页面的load事件触发时
- `domcontentloaded` - 页面的 `DOMContentLoaded` 事件触发时
- `networkidle0` - 不再有网络连接时触发（至少500毫秒后）
- `networkidle2` - 只有2个网络连接时触发（至少500毫秒后）

其他能力

在浏览器中手动可执行的绝大多数操作都可以使用 Puppeteer 来完成！

- 禁用 js
- 禁用图片
- 注入 js 文件或代码
- 离线模式
- 网络请求拦截
- 鼠标移动和操作
- 键盘输入
- 点击 某个元素
- 设置 UserAgent
- 对话框的操作
- ……

THANKS!

@衣明志： qihangnet@hotmail.com