# Playwright自动化总结

## 问题： 在使用Playwright进行网页自动化时，页面资源未完全加载即进行截图操作。

# 解决：

# 使用wait\_for\_load\_state等待页面达到特定的加载状态，例如，networkidle 表示网络请求基本静止，domcontentloaded 表示 DOM 完全加载，而 load 则表示页面及其所有资源都加载完毕。

# await self.page.wait\_for\_load\_state('networkidle')

# 运用wait\_for\_selector等待页面上特定元素的出现，表明页面的关键部分已加载。

# await self.page.wait\_for\_selector('#some-element-id')

# 通过wait\_for\_function执行JavaScript代码，检查页面上特定的状态或条件，确保满足截图前的页面状态。

# await self.page.wait\_for\_function('window.someGlobalVariable === true')

# 结合使用上述方法，先等待网络请求静止，再等待特定元素出现，确保截图时页面完全加载。

# 修改截图方法，加入等待逻辑，如等待networkidle和特定元素#page-loaded出现后，再执行截图操作。

## 页面加载超过了30秒的默认超时时间而报错

# 增加超时时间： await self.page.goto(self.url, timeout=60000)

# 检查URL和网络连接：

# 优化页面加载：尝试禁用一些可能影响页面加载速度的特性，比如JavaScript或图片加载，或者使用Playwright的setJavaScriptEnabled和setBypassCSP等方法。

# 禁用JavaScript：如果页面的主要内容在没有JavaScript的情况下也能加载，可以考虑暂时禁用它以加快加载速度。但是，如果你的目标页面依赖于JavaScript来加载内容，那么这个选项可能不可行。

# # 启动浏览器并禁用JavaScript

# self.browser = await self.playwright.chromium.launch(headless=True)

# # 创建一个没有JavaScript的上下文

# context = await self.browser.new\_context(javascript\_enabled=False)

# self.page = await context.new\_page()

# 禁用图片加载：图片通常是网页加载时间的大头，可以尝试禁用它们以减少加载时间。但是，这可能会影响页面的完整性和功能。

# # 禁止图片加载

# await self.page.set\_extra\_http\_headers({"Accept-Language": "en-US,en;q=0.9"})

# await self.page.route('\*\*/\*.{jpg,png}', lambda route: route.abort())

# 设置视口大小：你已经在代码中设置了视口大小，这是一个好习惯，因为它可以帮助Playwright更快地渲染页面。

# 使用无头模式：虽然你当前设置的是非无头模式（headless=False），但在生产环境中使用无头模式可以显著提高性能和加载速度，因为不需要图形界面。

# 优化网络请求：Playwright提供了拦截网络请求的功能，可以用来取消不必要的请求，或者等待关键资源加载完毕后再继续执行。

# 处理异常： 在调用goto方法时添加异常处理，以便在超时或网络错误发生时能够优雅地处理，而不是让程序崩溃。