实践指南之网页转PDF

上传日期：2021.03.29　　706

---------- 跟着小编一起实现将网页转成 PDF 的功能吧 ----------

一、背景

开发工作中，需要实现网页生成 PDF 的功能，生成的 PDF 需上传至服务端，将 PDF 地址作为参数请求外部接口，这个转换过程及转换后的 PDF 不需要在前端展示给用户。

二、技术选型

该功能不需要在前端展示给用户，为节省客户端资源，选择在服务端实现网页生成 PDF 的功能。

Puppeteer

Puppeteer 是一个 `Node` 库，它提供了高级 `API` 来通过 `DevTools` 协议控制 `Chrome` 或 `Chromium`。

在浏览器中手动执行的大多数操作都可以使用 `Puppeteer` 完成，比如：

- 生成页面的屏幕截图和 PDF；
- 爬取 `SPA` 并生成预渲染的内容（即 `SSR`）；
- 自动进行表单提交，UI 测试，键盘输入等；
- 创建最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` 中运行测试；
- 捕获时间线跟踪网站，以帮助诊断性能问题；
- 测试 `Chrome` 扩展程序。

从上可见，`Puppeteer` 可以实现在 `Node` 端生成页面的 PDF 功能。

三、实现步骤

安装

进入项目，安装 `puppeteer` 到本地。

```
$ npm install -g cnpm --
registry=https://registry.npm.taobao.org
$ cnpm i puppeteer --save
```

注意还有另一个版本的 `puppeteer`，就是 `puppeteer-core`，使用 `puppeteer-core` 需注意本地有可连接的浏览器，且安装的 `puppeteer-core` 版本与打算连接的浏览器兼容。连接本地浏览器方法如下：

```
const browser = await puppeteer.launch({
  executablePath: '/path/to/Chrome'
});
```

本项目需要部署至服务端，因此选择安装的是 `puppeteer`。

生成浏览器

```
const browser = await puppeteer.launch({
    headless: true,
    args: ['--no-sandbox', '--font-render-
hinting=medium']
  })
```

`headless` 代表无头模式，生成的浏览器在后端打开，前端不会有展示。

> *小建议：本地调试时，建议设置 `headless: false`，可以启动完整版本的浏览器，直接在浏览器窗口查看内容。*

打开新页面

生成浏览器后，在浏览器中打开新页面。

```
const page = await browser.newPage()
```

跳转到指定页面

跳转至要生成 PDF 的页面。

```
await page.goto(`${baseURL}/article/${id}`, {
    timeout: 60000,
    waitUntil: 'networkidle2', // networkidle2
会一直等待，直到页面加载后不存在 2 个以上的资源请求，这
种状态持续至少 500 ms
  })
```

`timeout` 是最长的加载时间，默认 30s，网页加载时间长的情况下，建议将 `timeout` 值改大，防止超时报错。

`waitUntil` 表示页面加载到什么程度可以开始生成 PDF 或其他操作了，当网页需加载的图片资源较多时，建议设置为 `networkidle2`，有以下值可选：

- load：当 `load` 事件触发时；

- domcontentloaded： 当 `DOMContentLoaded` 事件触发时；
- networkidle0： 页面加载后不存在 0 个以上的资源请求，这种状态持续至少 500 ms；
- networkidle2： 页面加载后不存在 2 个以上的资源请求，这种状态持续至少 500 ms。

指定路径，生成pdf

上述指定的页面加载完成后，将该页面生成 PDF。

```
  const ext = '.pdf'
  const key = randomFilename(title, ext)
  const _path = path.resolve(config.uploadDir, key)
  await page.pdf({ path: _path, format: 'a4' })
```

`path` 表示将 PDF 保存到的文件路径，如果未提供路径，PDF 将不会保存至磁盘。

> *小建议：不管 PDF 是不是需要保存到本地，建议在调试的时候都设置一个path，方便查看生成的 PDF 的样式，检查是否有问题。*

`format` 表示 PDF 的纸张格式，a4 尺寸为 8.27 英寸 x 11.7 英寸，是传统的打印尺寸。

> *注意：目前仅支持headless: true 无头模式下生成 PDF*

关闭浏览器

所有操作完成后，关闭浏览器，节约性能。

```
  await browser.close()
```

四、难点

图片懒加载

由于需生成 PDF 的页面是文章类型的页面，包含大量图片，且图片引入了懒加载，导致生成的 PDF 会带有很多懒加载兜底图，效果如下图：

解决方法是跳转到页面后，将页面滚动到底部，所有图片资源都会得到请求， `waitUntil` 设置为 `networkidle2` ，图片就能加载成功了。

```
await autoScroll(page) // 因为文章图片引入了懒加载，所以需要把页面滑动到最底部，保证所有图片都加载出来

/**
 * 控制页面自动滚动
 * */
function autoScroll (page) {
  return page.evaluate(() => {
    return new Promise<void>(resolve => {
      let totalHeight = 0
      const distance = 100
      // 每200毫秒让页面下滑100像素的距离
      const timer = setInterval(() => {
        const scrollHeight = document.body.scrollHeight
        window.scrollBy(0, distance)
        totalHeight += distance
        if (totalHeight >= scrollHeight) {
          clearInterval(timer)
          resolve()
        }
      }, 200)
    })
  })
}
```

这里用到了 `page.evaluate()` 方法，用来控制页面操作，比如使用内置的 `DOM` 选择器、使用 `window` 方法等等。

CSS 打印样式

根据 官网 说明， `page.pdf()` 生成 PDF 文件的样式是通过 `print css media` 指定的，因此可以通过 `css` 来修改生成的 PDF 的样式，以本文需求为例，生成的 PDF 需要隐藏头部、底部，以及其他和文章主体无关的部分，代码如下：

```
@media print {
  .other_info,
  .authors,
  .textDetail_comment,
  .detail_recTitle,
  .detail_rec,
  .SuspensePanel {
    display: none !important;
  }

  .Footer,
  .HeaderSuctionTop {
    display: none;
  }
}
```

登录态

由于存在一部分文章不对外部用户公开，需要鉴权用户身份，符合要求的用户才能看到文章内容，因此跳转到指定文章页后，需要在生成的浏览器窗口中注入登录态，符合条件的登录用户才能看到这部分文章的内容。

采用注入 `cookie` 的方式来获取登录态，使用 `page.evaluate()` 设置 `cookie` ，代码如下：

```
async function simulateLogin (page, cookies, domain) {
  return await page.evaluate((sig, sess, domain) => {
    let date = new Date()
    date = new Date(date.setDate(date.getDate() + 1))
    let expires = ''
    expires = `; expires=${date.toUTCString()}`
    document.cookie = `koa:sess.sig=${sig}${expires}; domain=${domain}; path=/`
    document.cookie = `koa:sess=${sess}=${expires}; domain=${domain}; path=/  //'=是这个cookie的value
    document.cookie = `is_login=true${expires}; domain=${domain}; path=/`
  }, cookies['koa:sess.sig'], cookies['koa:sess'], domain)
```

```
}

await simulateLogin(page, cookies,
config.domain.split('//')[1])
```

> *小建议：Puppeteer 也有自带的 api 实现 cookie 注入，如 page.setCookie({name: name, value: value})，但是我用这个方式注入没能获取到登录态，没有找到具体原因，建议还是直接用我上面这个方法来注入 cookie，注意除 name 和 value 外，expires、domain、path 也需要配置。*

Docker 部署 Puppeteer

根据上文操作，本地已经可以成功将页面生成 PDF 了，本地体验没问题后，需要部署到服务端给到测试、上线。

没有修改 `Dockerfile` 时，部署后发现了如下错误：

官网有给 Docker 配置说明 可以参考，最终实践可用的 `ubuntu` 系统的 `Dockerfile` 如下：

```
FROM  is.jd.com/o2athena/ubuntu-nvm-multimedia

ENV TIMEZONE=Asia/Shanghai
ENV NODE_PATH=/usr/lib/node_modules
ENV
PUPPETEER_DOWNLOAD_HOST=https://storage.google
apis.com.cnpmjs.org

# 复制仓库到 /app
# node_modules/* 和 /web/* 不会复制
WORKDIR /app

COPY . .

# 安装 puppeteer 依赖
RUN apt-get update && \
    apt-get install -y libgbm-dev && \
    apt-get install gconf-service libasound2
libatk1.0-0 libatk-bridge2.0-0 libc6 libcairo2
libcups2 libdbus-1-3 libexpat1 libfontconfig1
libgcc1 libgconf-2-4 libgdk-pixbuf2.0-0
libglib2.0-0 libgtk-3-0 libnspr4 libpango-1.0-
0 libpangocairo-1.0-0 libstdc++6 libx11-6
libx11-xcb1 libxcb1 libxcomposite1 libxcursor1
libxdamage1 libxext6 libxfixes3 libxi6
libxrandr2 libxrender1 libxss1 libxtst6 ca-
certificates fonts-liberation libappindicator1
libnss3 lsb-release xdg-utils wget build-
essential libcairo2-dev libpango1.0-dev
libjpeg-dev libgif-dev librsvg2-dev -y && \
    apt-get install -y fonts-ipafont-gothic
fonts-wqy-zenhei fonts-thai-tlwg fonts-kacst
fonts-freefont-ttf --no-install-recommends

RUN ln -snf /usr/share/zoneinfo/$TIMEZONE
/etc/localtime \
  && echo $TIMEZONE > /etc/timezone \
  && apt-get update \
  && apt-get install -y tzdata

# 安装依赖
RUN . $NVM_DIR/nvm.sh \
  && nvm install 13 \
  && nvm use 13 \
  && npm install --production --
registry=http://registry.m.jd.com

ENTRYPOINT . $NVM_DIR/nvm.sh \
  && nvm use 13 \
  && chmod +x /app/start.sh \
  && /app/start.sh \
  && sleep 9999d
```

只需要重点关注 **安装 puppeteer 依赖** 部分即可。

> *注意：在 v1.18.1 之前，Puppeteer 至少需要 Node v6.4.0。从 v1.18.1 到 v2.1.0 的版本都依赖于 Node 8.9.0+。从 v3.0.0 开始，Puppeteer 开始依赖于 Node 10.18.1+。配置 Dockerfile 时也需要注意服务端的 node 版本。*

五、总结

本文讲述了实现在 `Node` 端将网页生成 PDF 文件的完整过程，总结为以下 3 点：

1. 技术选型，根据需求场景选择合适的手段实现功能；
2. 阅读 官方文档，快速过一遍文档才能少遇到些坑；
3. 破解难点，使用一个未使用的工具，会遇到没有解决过的难题，遇招拆招吧 ^ ^。

参照 Demo 源码 可快速上手上述功能，希望本文能对你有所帮助，感谢阅读❤️

· 往期精彩 ·

【直播回顾 · 程序媛的成长蜕变】

【大规格文件的上传优化】

【JDR DESIGN 开发小结】