上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

午睡无忧——针对 Canvas 的自动签到插件

522031910571-朱彦桥
522031910557-冯海桐

2025 年 2 月 8 日

1 项目背景

在大学生活中，有些在 12:55 开始的课程是令人感到困倦的，这让同学们失去了午睡的时间。当同学们想翘课在寝室里午睡的时候，又对老师是否会放签到码而感到担忧，往往课间就得起床查看一下。基于这点，我们开发了能检测签到码，并自动扫码签到的 Chrome 插件——午睡无忧，希望能让同学们睡个好觉。

项目主要分为两大部分：第一部分是前端视频流抓取，主要目的是从 Canvas 网站上捕获课程直播的链接，由冯海桐来完成；第二部分是后端视频流监控，主要目的是在服务器上实时监控直播画面，当检测到签到二维码时进行自动签到，由朱彦桥来完成。

顶层的架构如1所示。整体逻辑为，插件的 popup 部分和用户进行交互，进入一个带有视频的页面时，点开插件，popup 向 background 请求数据，background 返还视频元素的相关数据后，popup 渲染页面，然后如果想要进一步监控某一个直播流，点击 “开始监控”，和远端的服务器建立 websocket 联系，在服务器上启动监控任务。

2 实现过程

本章的每一小节都会尽量结合代码去讲解实现的逻辑，复杂处会借助思维导图。力求清楚地展示我们的工作和思路。

2.1 前端视频流抓取

抓取视频流的功能我们主要参考了“猫抓”(cat-catch) 这一插件。猫抓是一款能够嗅探浏览器页面资源的插件，提供资源解析和下载等多种功能，其源码链接为`https://github.com/xifangczy/cat-catch`。我们对此插件进行正向学习原理的同时，也逆向进行消融实验，提取主要功能，最后成功实现抓取视频流。

2.1.1 Chrome 插件结构

一个基本的 Chrome 插件的基本结构如图2所示。其中 manifest.json 是 Chrome 扩展程序的配置文件，定义了插件的基本信息和权限，声明了插件的脚本和界面元素，从而使其能在浏览器中正常运行。js 文件夹中是插件的主要功能实现，分为后台 background.js 和前台 popup.js。后台主要实现对

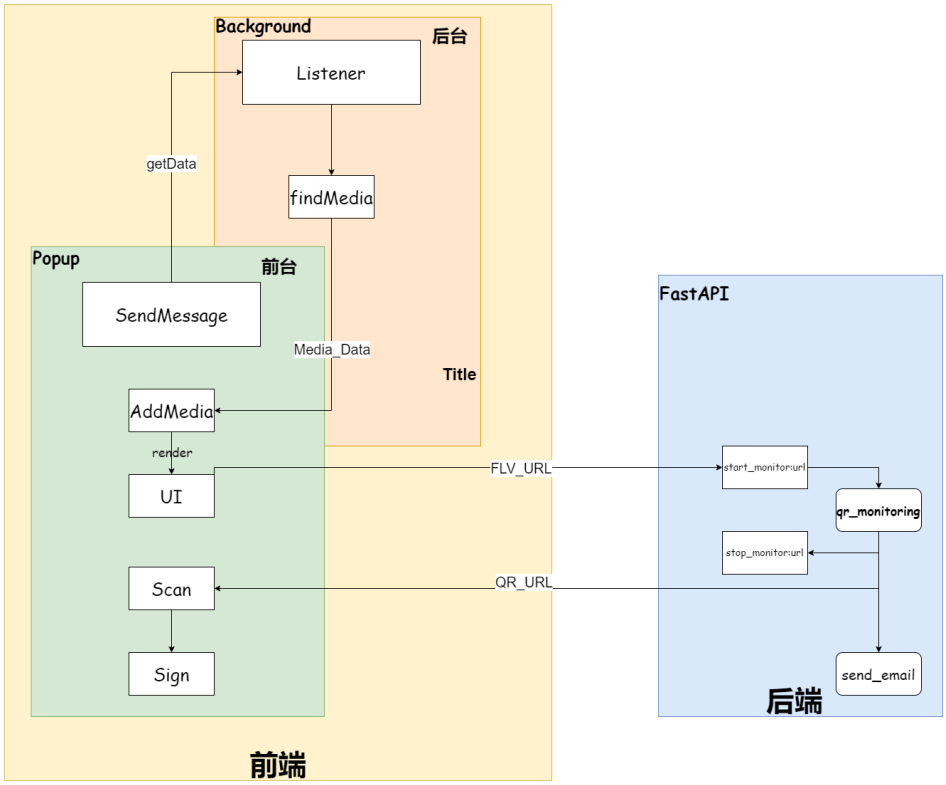

图 1 顶层架构图

浏览器发送请求和接收回复的捕获与相关事件处理，并将目标的数据发送给前台；前台一方面监听后台发送的数据，另一方面要实时处理插件弹出页面 popup.html 的事件，如鼠标点击按钮、敲击键盘等。css 文件夹下的 CSS 文件主要用于定义 HTML 文档的样式和布局，它们通过选择器和属性来控制网页元素的外观。lib 文件下是其他的相关库。此外，如果考虑国际化和其他语言的匹配，还应该有 _locales 文件夹来处理信息的不同语言版本。

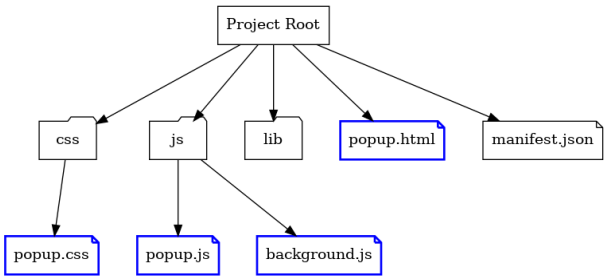

图 2 Chrome 插件基本结构

2.1.2 chrome.webRequest

使用 chrome.webRequest API 可观察和分析流量，以及拦截、阻止或修改传输中的请求，必须在 manifest.json 中声明 “webRequest” 权限，才能使用相关功能。要拦截子资源请求，插件必须同时有

权访问请求的网址及其发起者。一个成功的事件请求生命周期如图3所示。

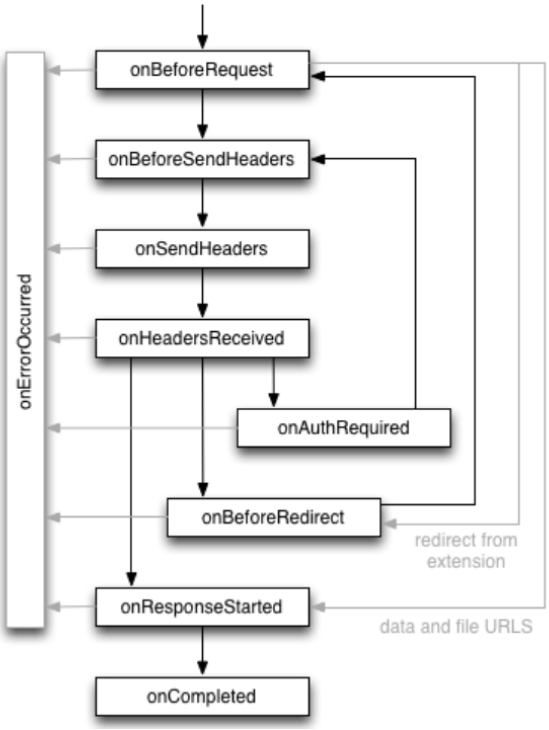

图 3 事件请求生命周期

onBeforeRequest 在请求即将发生时触发，此事件在建立任何 TCP 连接之前发送，可以用于取消或重定向请求。onBeforeSendHeaders 在请求即将发生且初始标头准备就绪时触发，该事件允许扩展程序添加、修改和删除请求标头。onSendHeaders 在所有扩展程序都完成修改请求标头后触发，并显示为修改后最终版本，不允许修改或取消请求。onHeadersReceived 在每次收到 HTTP(S) 响应标头时触发。onAuthRequired 在请求需要对用户进行身份验证时触发，该事件可以提供身份验证凭据。onBeforeRedirect 在即将执行重定向时触发。onResponseStarted 在收到响应正文的第一个字节时触发。onCompleted 在成功处理请求时触发。onErrorOccurred 在无法成功处理请求时触发。

Web 请求 API 可保证对于每个请求，onCompleted 或 onErrorOccurred 作为最终事件触发。

后台对于视频流的抓取，主要在 onResponseStarted 这一步完成，示例代码如下：

```
chrome.webRequest.onResponseStarted.addListener(
    function (data) {
        //对data的处理
        } catch (e) { console.log(e, data); }
    }, { urls: ["<all_urls>"] }, ["responseHeaders"]
);
```

其中，获取的数据 data 结构如下：

```
{
  frameId: integer,
  initiator: string,
```

```
  method: string,
  parentFrameId: integer,
  requestId: string,
  statusCode: integer,
  statusLine: string,
  tabId: integer,
  timeStamp: double,
  type: string,
  url: string,
  responseHeaders: array of object
}
```

其中 url 一项是我们关心的，它可能包含了直播流的链接，但还需要进一步判断。

由于本插件是针对 Canvas 这单一网站开发的，所以理应在 onBeforeRequest 使对当前的网站进行正则匹配，如果不是 Canvas，插件不激活。但是可以在 Canvas 上测试的直播样本过少（并非每一个老师都进行扫码签到），于是为了方便测试，也考虑到本插件可以扩展应用到其他直播的检测，所以删除了这一部分。

2.1.3 视频流资源的判断

上一步获得的 url，以 bilibili 直播为例，如下所示：

`https://cn-sh-fx-01-03.bilivideo.com/live-bvc/114514/live_2867_64485_minihevc/index.m3u8?expires=1735835869&len=0&oi=3396864110&pt=web&qn=250&trid=10071045ffff4e3ef65026546d27d6776b2&sigparams=cdn,expires,len,oi,pt,qn,trid`

url 中有大串的查询参数，不过我们只关心路径最后的文件扩展名，bilibili 直播的视频流类型为 m3u8，而 Canvas 直播的视频流类型为 flv。现在对文件扩展名进行判断与筛选，具体代码如下：

```
const urlParsing = new URL(data.url);
let [name, ext] = fileNameParse(urlParsing.pathname);


function fileNameParse(pathname) {
    // 解码 URI 组件，获取文件名
    let fileName = decodeURI(pathname.split("/").pop());
    // 分割文件名，获取扩展名
    let ext = fileName.split(".");
    // 如果没有扩展名，设置 ext 为 undefined，否则获取扩展名并转换为小写
    ext = ext.length == 1 ? undefined : ext.pop().toLowerCase();
    // 返回文件名和扩展名（如果有）
    return [fileName, ext ? ext : undefined];
}
```

其中 URL() 函数将用于解析和处理 url 字符串。它将一个 url 字符串解析为一个 url 对象，并提供属性和方法来访问和操作 url 的各个部分。

后台筛选出需要的扩展名，并将符合要求的原数据 data 通过 chrome.runtime.sendMessage() 发送

给前台。

2.1.4 前台页面

前台页面的主体较为简单，如下所示：

```
<body>
  <div id="mediaList" class="container hide TabShow"></div>
  <script src="js/popup.js"></script>
</body>
```

mediaList 包含了当前界面所有的视频流，因为 Canvas 存在两个直播界面——教室摄像头和电脑屏幕。有的老师直接用教室电脑上课，所以检测电脑屏幕的视频流即可；而有的老师自带电脑，这就需要检测教室摄像头拍到的大屏幕画面。

mediaList 仅包含视频流的省略讯息，具体的 url 和检测选项需要点开详情界面，最终效果如图4所示。

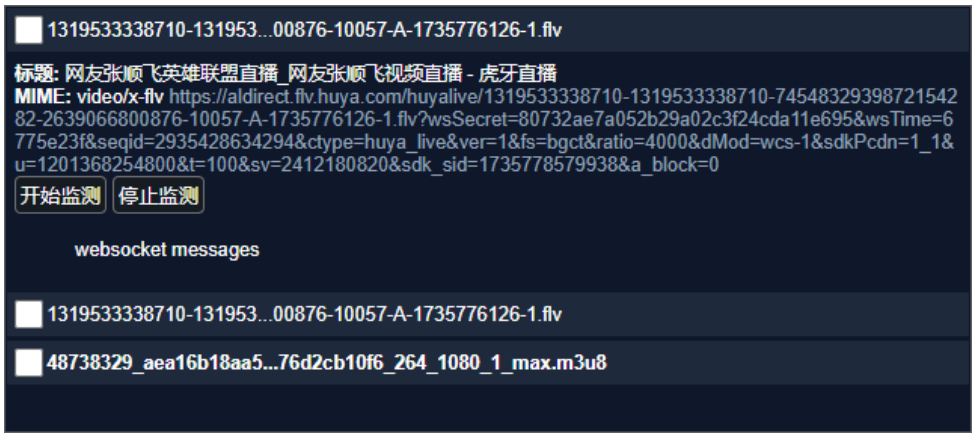

图 4 前台页面最终效果

2.2 后端视频流监控

2.2.1 前端添加按钮和消息列表

这部分主要在 popup.js 中完成，不在 background.js 里面实现主要是不想把这个插件做得过于成熟，被人滥用。

首先是简单的按钮加入（省去了 css 的定义）：

```
<div>
    <button id="start-monitor">开 始 监 测</button>
    <button id="stop-monitor">停 止 监 测</button>
</div>
<ul id="messages">websocket messages
</ul>
```

然后为开始监测和停止监测绑定按钮绑定函数，这里仅展示代码逻辑，用注释代替实现这一功能的代码。

```
// 绑定 start-monitor 按钮事件
data.html.find("#start-monitor").click(function () {
    // 连接服务器并且发送start_monitor命令
    const socket = new WebSocket("ws://8.137.110.236:8000/ws");
    data.socket = socket;
    socket.onopen = function () {
        console.log("socket open");
        socket.send(`start_monitor:${data.url}`);
    };

    // websocket 等待，需要一直接受消息
    socket.onmessage = function (event) {
        if (event.data.includes("https://mlearning.sjtu.edu.cn")) {
            // 使用正则表达式提取URL
            if (match && match[0]) {
                const url = match[0];  // 提取的URL
                console.log("提取的URL: ", url);
                // 解析原始URL
                // 加上学生id，得到newUrl
                window.location.href = newUrl;
                // 发送停止消息
                data.socket.send(`stop_monitor:${data.url}`);
        }
    }
});

// 绑定 stop-monitor 按钮事件
data.html.find("#stop-monitor").click(function () {
    // 发送停止消息
    data.socket.send(`stop_monitor:${data.url}`);
});
```

这里有必要强调从原始 URL 到 newURL 的过程，在实践时发现，原始 URL 在电脑端直接访问是 404，而通过手机扫码得到的链接却是另外一个。

- 原始 URL：`https://mlearning.sjtu.edu.cn/lms/mobile/forscan/{args}`
- 新 URL：`https://mlearning.sjtu.edu.cn/lms/mobile/?t=1618827503000/#/pages/course/CourseDetailPage/{args}`

新 URL 涉及到 SPA 应用，经过查阅，SPA 是一种特殊的 Web 应用，是加载单个 HTML 页面并在用户与应用程序交互时动态更新该页面的。它将所有的活动局限于一个 Web 页面中，仅在该 Web 页面初始化时加载相应的 HTML。JavaScript、Css。一旦页面加载完成，SPA 不会因为用户的操作而进行页面的重新加载或跳转，而是利用 JavaScript 动态的变换 HTML (采用的是 div 切换显示和隐藏)，从而实现 UI 与用户的交互。在 SPA 应用中，应用加载之后就不会再有整页刷新。相反，展示逻辑预先加载，并有赖于内容 Region (区域) 中的视图切换来展示内容。

在电脑端直接访问原始 URL 时，浏览器会向服务器请求具体的路径 /forscan/。如果服务器未配置对应的路由处理，服务器无法找到该资源，导致 404 错误。而通过手机扫码访问时，生成的新 URL 利用了 SPA 的路由机制，浏览器只向服务器请求基础路径，服务器返回入口页面，前端框架再根据 # 后的部分渲染相应的页面。

2.2.2 websocket

当前端想要启动监控任务的时候，和服务器建立联系。我们先阐明服务器端的接口。

首先是接口 ws 的逻辑，这里我们用了 FastAPI 进行封装。在 FastAPI 中，WebSocket 路由的定义是通过 @app. 这样的修饰器来让 FastAPI 封装这个服务。

```
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await manager.connect(websocket)
    try:
        while True:
            data = await websocket.receive_text()  # 接收客户端发来的消息
            print(f"收到来自客户端的消息:␣{data}")
            # data去掉前后空格
            data = data.strip()
            if data.startswith("start_monitor:"):
                # 启动监控
                # url = data.split(":")[1]
                url = data.split(":",1)[1].strip()
                await manager.send_message(f"服务端得到的url:␣{url}")
                if url in monitored_streams:
                    await manager.send_message(f"该URL已经在监控中:␣{url}")
                else:
                    # 启动监控任务
                    task = asyncio.create_task(qr_monitoring(url))
                    monitored_streams[url] = task
                    await manager.send_message(f"已开始监控␣URL:␣{url}")

            elif data.startswith("stop_monitor:"):
                # 停止监控
                url = data.split(":",1)[1].strip()
                if url not in monitored_streams:
                    await manager.send_message(f"该URL没有在监控中:␣{url}")
                else:
                    task = monitored_streams.pop(url)
                    task.cancel()
    except WebSocketDisconnect:
        print("客户端断开连接")
        manager.disconnect(websocket)
```

然后是 API_KEY 的机制, 可以防止未授权的同学直接使用。

```
# API Key 认证依赖
```

```
def get_api_key(api_key: str):
    if api_key != API_KEY:
        raise HTTPException(status_code=403, detail="Could not validate credentials
            ")
    return api_key
```

2.2.3 后端监控

这部分重点展示 qr_monitoring 函数的逻辑

```
import cv2
import pyzbar.pyzbar as pyzbar
import asyncio
import time
async def qr_monitoring(url):
    cap = cv2.VideoCapture(url)
    # 怎么确保实时性？设置缓冲为0，这样每次调用ccap.read()才返回的是最新帧
    cap.set(cv2.CAP_PROP_BUFFERSIZE, 0)
    # https://stackoverflow.com/a/54755738/25110723
    if not cap.isOpened():
        # 发送异常消息
    # 初始变量定义
    email_sent = False ,cooldown_time = 0.1 ,last_sent_time = 0 ,last_process_time
        = 0
    cnt = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            # 发送异常消息
        current_time = time.time()
        # 每 cooldown_time 秒处理一次帧
        if current_time - last_process_time >= cooldown_time:
            last_process_time = current_time
            qr_codes = pyzbar.decode(frame)
            if qr_codes:
                await manager.send_message(body) # 匹配则通过websocket发送消息
                send_email(subject, body, NOTIFY_EMAIL) # 并且调用发送邮件函数
                # 冷却机制
                if email_sent:
                    cooldown_time = 10
                else:
                    email_sent = False  # 重置邮件发送状态
                    cooldown_time = 0.1  # 恢复默认冷却时间
    cap.release()
```

为了阐释这个核心函数的逻辑，图5通过思维导图的方式再现了一遍。

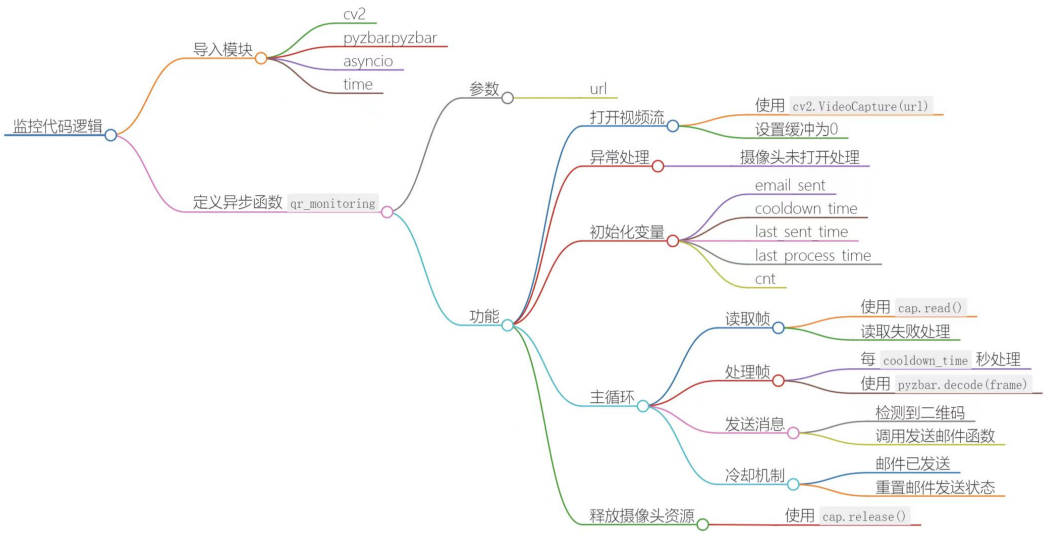

图 5 qr_monitoring 函数思路图

2.2.4 邮件发送

```python
# 邮件发送函数
def send_email(subject, email_body, to_email=NOTIFY_EMAIL):
    # 创建邮件
    msg = MIMEMultipart()
    msg['From'] = SMTP_USERNAME
    msg['To'] = to_email
    msg['Subject'] = subject

    # 邮件正文
    msg.attach(MIMEText(email_body, 'plain'))

    # 连接到SMTP服务器并发送邮件
    try:
        server = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
        server.starttls()  # 启用TLS加密
        server.login(SMTP_USERNAME, SMTP_PASSWORD)
        text = msg.as_string()
        server.sendmail(SMTP_USERNAME, to_email, text)
        server.quit()
        print("Email sent successfully!")
    except Exception as e:
        print(f"Failed to send email: {e}")
```

3 项目成果

3.1 插件演示

插件演示视频已经上传到 bilibili，链接为https://www.bilibili.com/video/BV1fj6iYuE6Q/。我们用电脑模拟课堂直播，筛选了无关的二维码并成功进行了自动签到。与此同时，捕获到二维码的信息会立刻发送给用户的邮箱，如图6所示。

图 6 邮件信息

另外，我们还在 github 上建立了此项目的仓库：https://github.com/fhtcb/TakeANap。

3.2 创新性与可扩展性

本项目在设计与实现过程中注重创新性和可扩展性的结合，确保插件不仅满足当前的需求，还具备良好的适应性以应对未来的扩展和优化。以下是本项目的主要创新点和可扩展性分析：

3.2.1 创新性

- **自动二维码检测与签到**：传统的签到方式依赖于学生手动输入签到码或扫描二维码，存在操作繁琐和遗漏的风险。本插件通过实时监控直播视频流，自动检测签到二维码并完成签到，大大简化了用户操作，提高了签到的效率和准确性。
- **前后端分离的架构设计**：插件采用前后端分离的 C/S 架构，前端负责与用户交互和视频流的抓

取，后端负责视频流的实时监控和二维码识别。这种设计不仅提高了系统的模块化程度，便于维护和升级，还增强了系统的安全性，避免了敏感信息的泄露。

- **实时通讯与任务管理**: 利用 FastAPI 和 WebSocket 技术实现前后端的实时通讯，确保监控任务的及时响应和结果反馈。此外，通过任务管理器动态启动和停止监控任务，提高了系统资源的利用效率。
- **邮件通知功能**: 在检测到签到二维码后，系统不仅完成自动签到，还通过 SMTP 协议发送邮件即时通知用户签到结果。这一功能增强了用户体验，使用户能够及时了解签到情况，避免因系统故障或其他原因导致的签到失败。
- **安全性设计**: 插件在通信过程中采用 API Key 机制，防止未授权的访问和操作。同时，插件对用户的个人 Cookies 进行了严格保护，确保用户的隐私和数据安全。

3.2.2 可扩展性

- **多平台支持**: 虽然本插件初步针对 Canvas 平台开发，但其模块化的设计使其可以方便地扩展到其他在线教育平台或直播平台。通过调整视频流的抓取和二维码的识别逻辑，插件可以适应不同平台的签到机制。
- **功能扩展**: 插件的前后端分离架构允许开发者在现有基础上添加更多功能，特别是后端，完全不局限于扫码这一简单的交互。可以结合模式匹配的计算负载更大的功能，实现更复杂的任务，如视频内容分析、行为识别等，从而进一步提升插件的功能和应用范围。而这些集成只需要更改 API，对于前端用户来说是无痛切换的。
- **高并发支持**: 后端服务器采用异步编程和任务管理机制，可以高效地处理多个并发的监控任务。这使得系统能够支持大量用户同时使用，而不会显著影响性能。
- **易于维护与升级**: 项目的代码结构清晰，前后端逻辑分离，便于开发者进行维护和功能升级。新功能的添加或现有功能的优化可以在不影响整体系统稳定性的前提下进行。

4 个人收获和致谢

朱彦桥

这次计算机网络强迫我自己去动手实现自己的想法。前期讨论课题的时候，我们犹豫不决，总想着用某某某工具实现一个更加 fancy 的功能，但最后，我们选择了相对朴素的自动二维码签到。一个是因为这个最符合我们的实际需求，另一个则是因为我们想真正地有了钉子再去找锤子。

项目整体架构简单，但是过程却不容易，由于缺失相关经验，我们必须先定义好功能，再想方设法地去实现，一方面，清晰易懂的蓝图和实用方便的功能经常让我心潮澎湃，另一方面，有时一个简单的小功能就能卡住一下午（比如 cv 如何读取最新的一帧）。在这种交杂的心绪中，我们积极沟通，一步步实现我们的目标，虽然简陋，但确实很有成就感。过程中也学到一些小知识，比如 SPA, 单页面应用，比如 cv 库 read 的逻辑。

再结合这学期的内容，当我调用 websocket 的功能的时候，我脑中很难不浮现 TCP 的精妙设计，当我使用 html 调用相关的 js 的时候，并正确渲染出网页的时候，我无法不赞叹分层结构的伟力，感谢计算机网络这门课教授知识的同时，切切实实地让我体验了“站在巨人肩膀”上那种热血和感动。

冯海桐

关于项目分工，我们本来以为我的部分较为困难，朱同学的部分较为简单，结果实际上手之后却发现恰恰相反。市面上已经有很多抓取网页资源的工具，猫抓只是其中之一，不过它确实给我提供了很大的帮助。

我在正向学习插件开发的时候经历了很多困难，比如显示乱码，比如无法运行。猫抓的代码库很大，功能很多，我完全不知道我想参考的代码在哪，学习代码如同大海捞针。我所在的科研组的薛家奇学长给了我启发，告诉我可以用消融实验的办法来提取想要的功能。因此，我开始大量的删除文件，删除函数，不断测试，抽丝剥茧，最后找到了大海里面的那一根针。这根针真的非常小巧，精妙，简单。

特别感谢朱同学的鼎力相助，实现了项目最核心的部分，并且包容了我的拖延症。