午睡无忧

——针对Canvas的自动签到插件

朱彦桥 冯海桐

目录

- 项目整体展示
- 前端视频流抓取
- 后端视频流监测
- 更多功能与QA

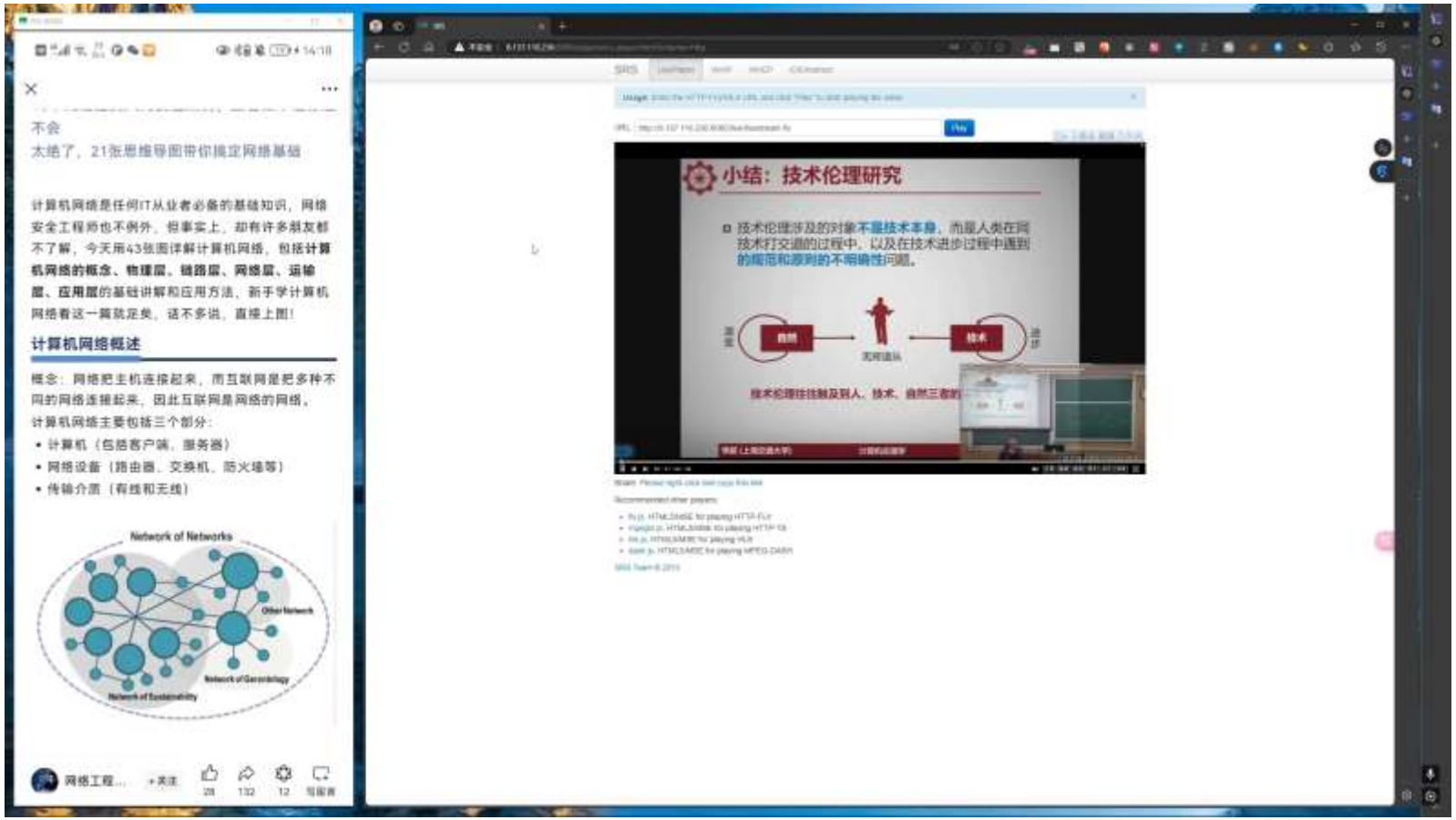
项目整体展示

立项背景

- 我们观看各种直播的时候，通常需要有这样的监控系统。当它出现某种事件的时候，能够触发警报，并执行下一步内容。
  - 比如，主播正在面向摄像头说话。
  - 比如，老师开始签到。

- 我们尝试开发了一款浏览器插件，来实现一键监控自动签到的功能。

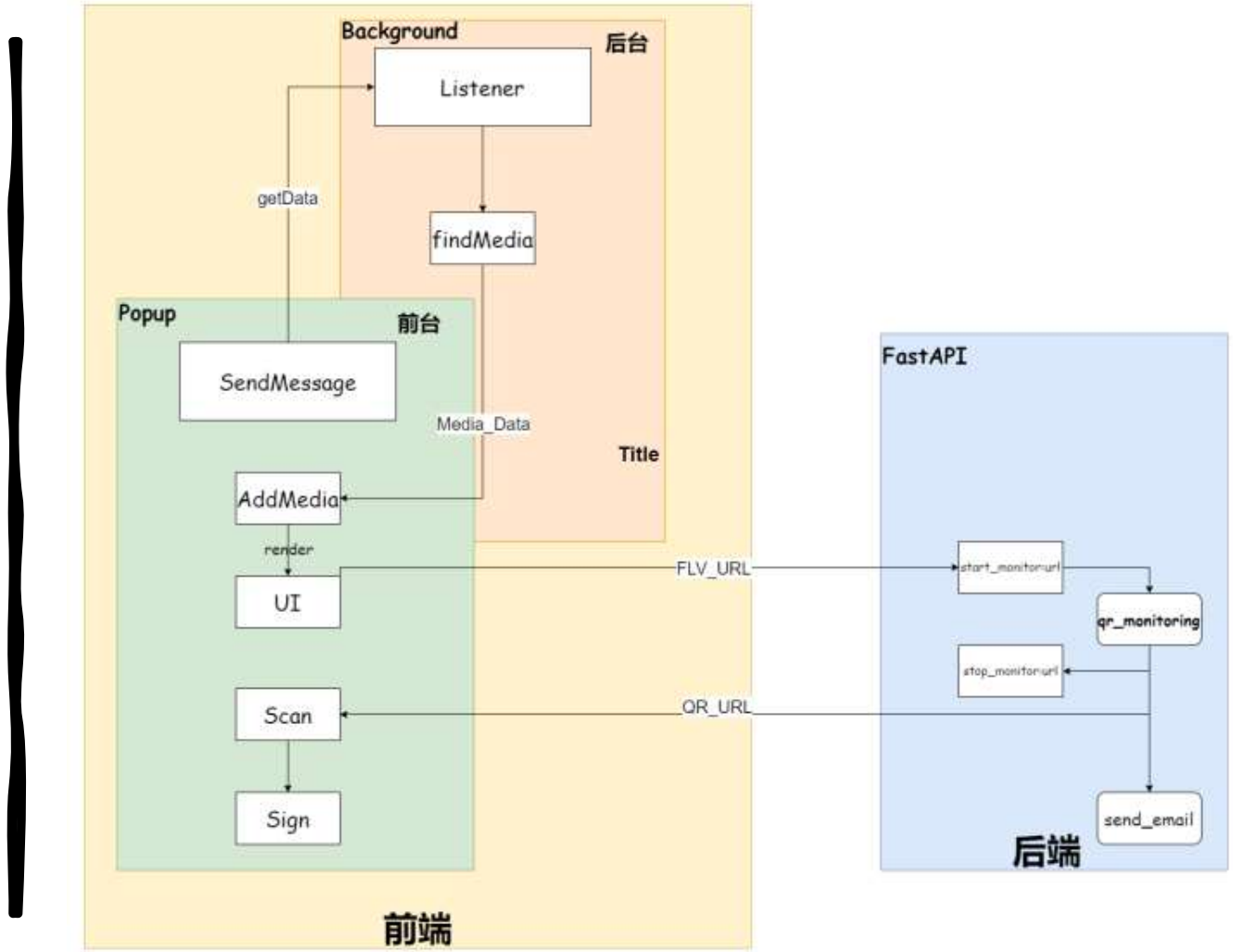
Demo展示


不会
太绝了，21张思维导图带你搞定网络基础
计算机网络是任何IT从业者必备的基础知识，网络安全工程师也不例外，但事实上，却有许多朋友都不了解，今天用43张图详解计算机网络，包括计算机网络的概念、物理层、链路层、网络层、运输层、应用层的基础讲解和应用方法，新手学计算机网络看这一篇就足矣，话不多说，直接上图！
计算机网络概述
概念：网络把主机连接起来，而互联网是把多种不同的网络连接起来，因此互联网是网络的网络。
计算机网络主要包括三个部分：
• 计算机（包括客户端、服务器）
• 网络设备（路由器、交换机、防火墙等）
• 传输介质（有线和无线）
Network of Networks
Other Network
Network of Gerontology
Network of Sustainability
网络工程...
+关注
28
132
12
小结：技术伦理研究
技术伦理涉及的对象不是技术本身，而是人类在同技术打交道的过程中，以及在技术进步过程中遇到的规范和原则的不明确性问题。
自然
技术
无何道场

C/S架构

- 如果是单纯的一个python，逐个遍历自己的课程，发现如果处于直播阶段，则开始监控
  - 自动签到需要利用个人的cookies
  - 我们的cookies过期是很快的，并不能奢望用户安装好一个应用，还要反复登录
- 所以和用户的交互最好是一个浏览器插件的形式
  - 但是监控涉及到一直运行程序，包括帧的分析和二维码的处理，所以最好放在一台服务器上进行
  - 如果我们需要扩展后续服务，不局限于检测简单二维码功能，处理程序可能对资源的要求更高，放在background.js里面负担也很重
- 分离用户的命令和对于命令的操作，用简单的C/S架构来分离

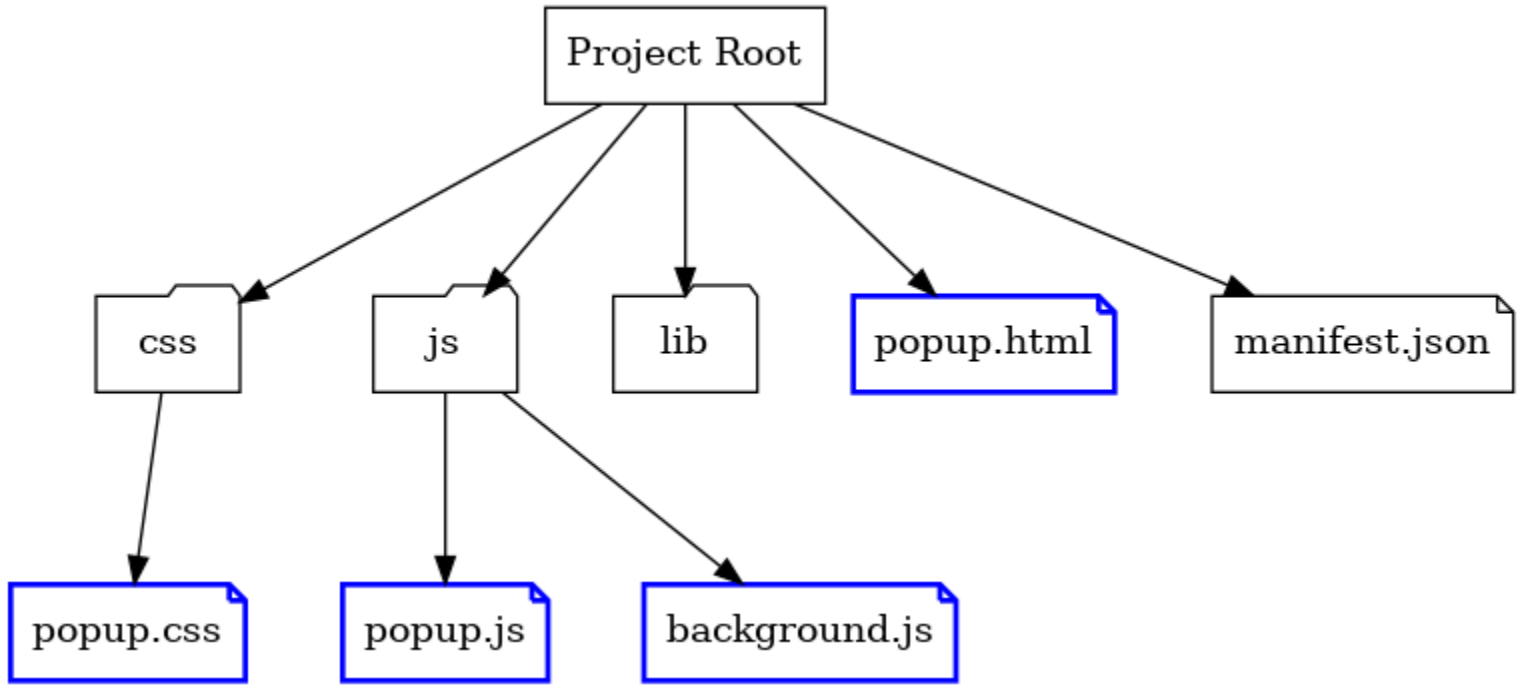
技术路线示意图


Background
后台
Listener
getData
findMedia
Popup
前台
SendMessage
Media_Data
Title
AddMedia
render
UI
Scan
Sign
前端
FastAPI
FLV_URL
start_monitor:url
qr_monitoring
stop_monitor:url
QR_URL
send_email
后端

前端视频流抓取

冯海桐

Chrome插件结构


Project Root
css
js
lib
popup.html
manifest.json
popup.css
popup.js
background.js

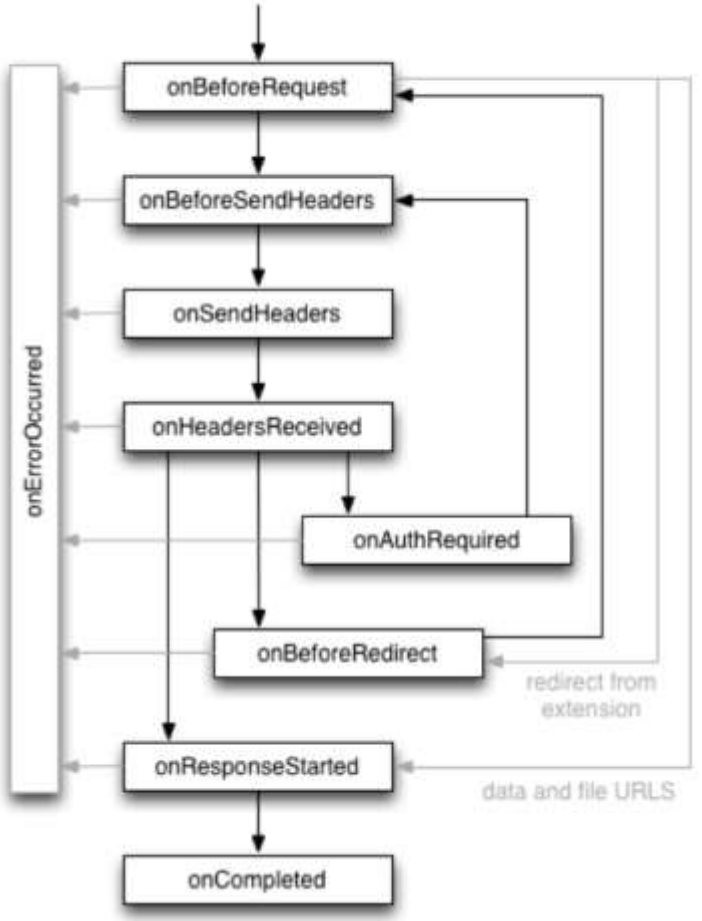
前端功能描述

- 后台background.js负责对网页发送的请求与收到的回应进行捕获与筛选，找到目标视频流发送到前台

- 前台popup.js负责监听后台发送的信息，并根据用户的指令，将相关信息发送给后端，并显示后端运行状态

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。



抓取视频流

- 利用 `chrome.webRequest.onResponseStarted.addListener()` 获取网络回应，其截获的数据结构如右所示：

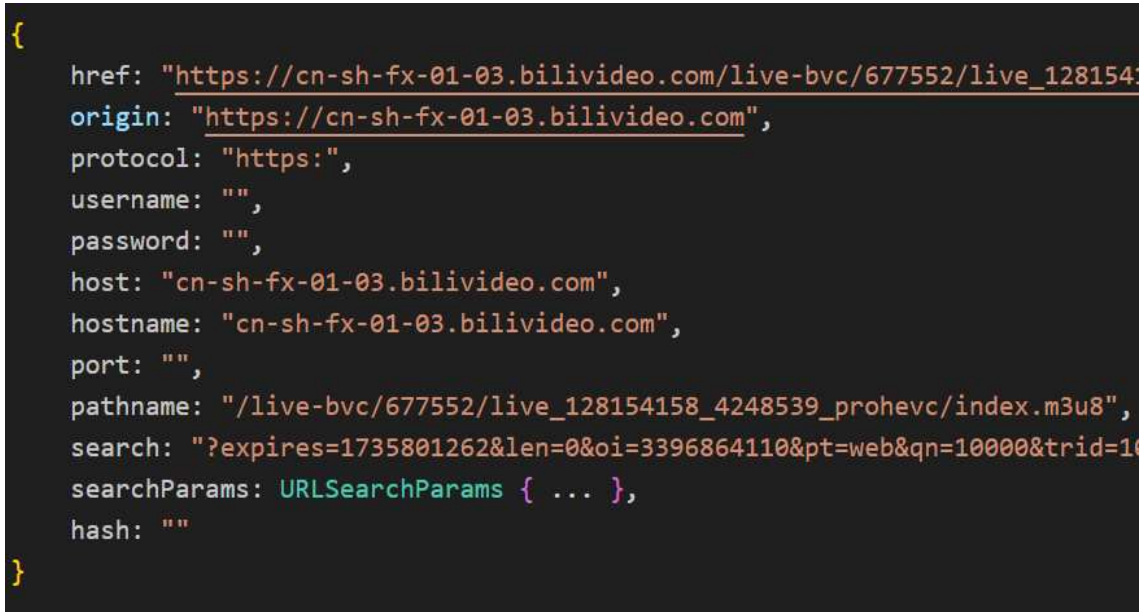
```
{
    "requestId": "string",
    "url": "string",
    "method": "string",
    "frameId": "integer",
    "parentFrameId": "integer",
    "tabId": "integer",
    "type": "string",
    "timeStamp": "number",
    "statusLine": "string",
    "statusCode": "integer",
    "responseHeaders": [
        {
            "name": "string",
            "value": "string"
        }
    ],
    "initiator": "string"
}
```

抓取视频流

- 利用URL()函数对获得的url分割，例如：

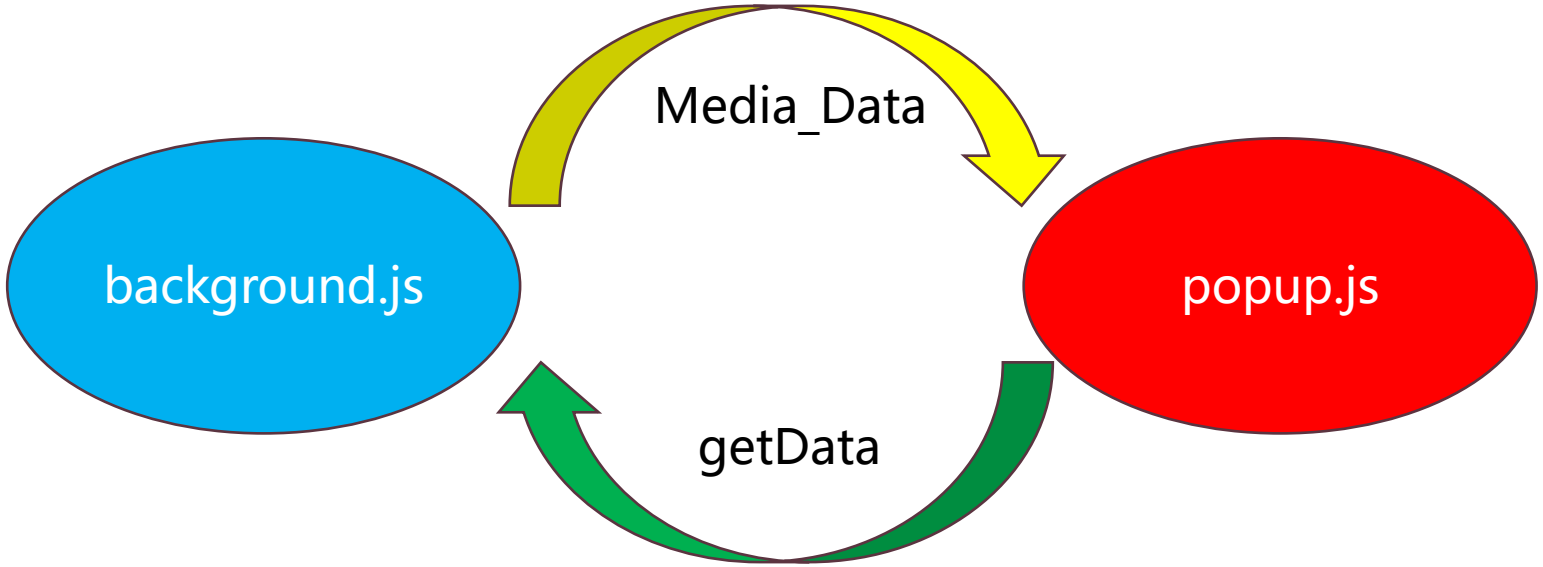
https://cn-sh-fx-01-03.bilivideo.com/live-bvc/677552/live_128154158_4248539_prohevc/index.m3u8?expires=1735801262&len=0&oi=3396864110&pt=web&qn=10000&trid=10077f0432cda0de1f5e1e4cab068067762b&sigparams=cdn,expires,len,oi,pt,qn,trid&cdn=cn-gotcha01&sign=daa5af503532fe4cc17f2a0a7e57b2d6&site=10d7a5b5788e18b5a91fd8020ce6e7a6&free_type=0&mid=450663095&sche=ban&bvchls=1&sid=cn-sh-fx-01-03&chash=1&bmt=1&sg=df&trace=5&isp=fx&rg=East&pv=Shanghai&qp=hv_10000&sk=657866af8fc16d8b2b84dfdde3e98143970e231f4ba7166220a3354568f1c7c7&suffix=prohevc&pp=srt&flvsk=1304f646dfeb4df8b6e7ff33c167d3ad74346564118ee355b45f1e0db5f3b4f0&origin_bitrate=965073&sl=5&deploy_env=prod&score=65&p2p_type=1&info_source=cache&hot_cdn=909513&source=puv3_onetier&vd=nc&zoneid_l=151355417&sid_l=live_128154158_4248539_prohevc&src=puv3&order=1

- 分割后得到：



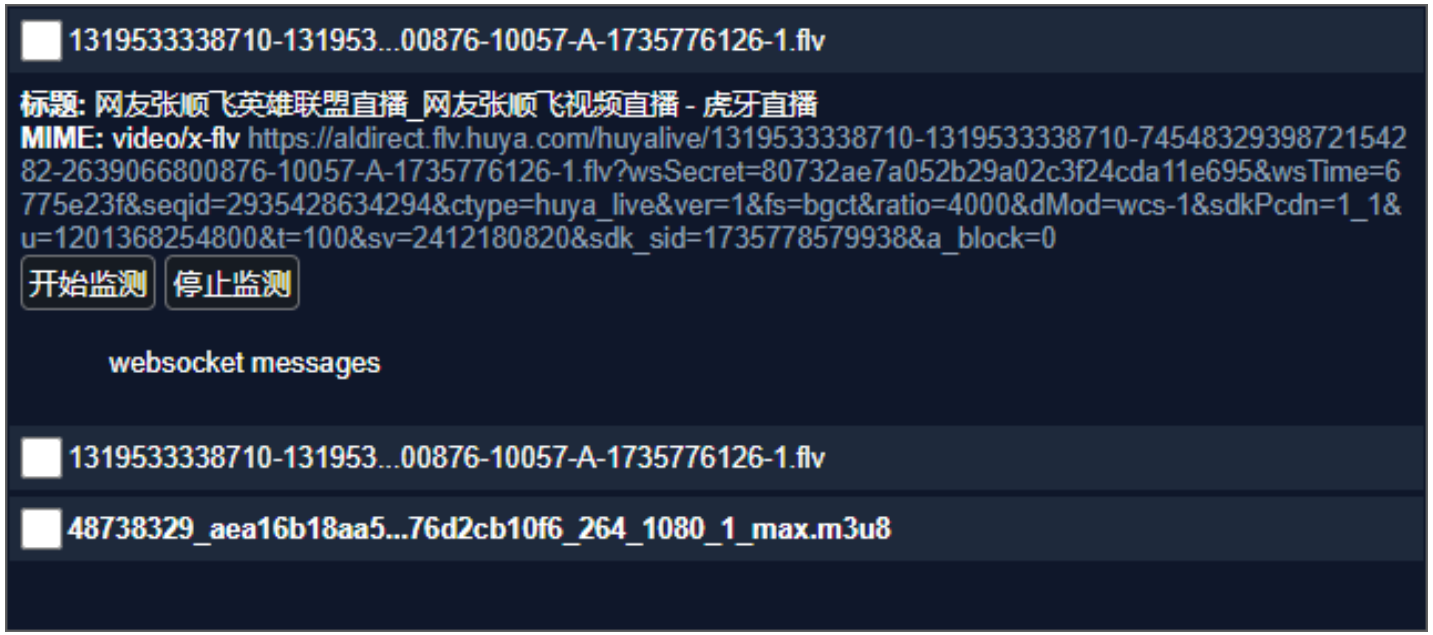
抓取视频流

- 筛选出有符合要求的后缀链接，如Canvas直播视频流文件后缀为flv，通过`chrome.runtime.sendMessage()`将对应的数据结构体发到前台。



前台页面

- 资源展示
- 命令按钮
- websocket message



后端视频流监测

朱彦桥

二维码动态参数

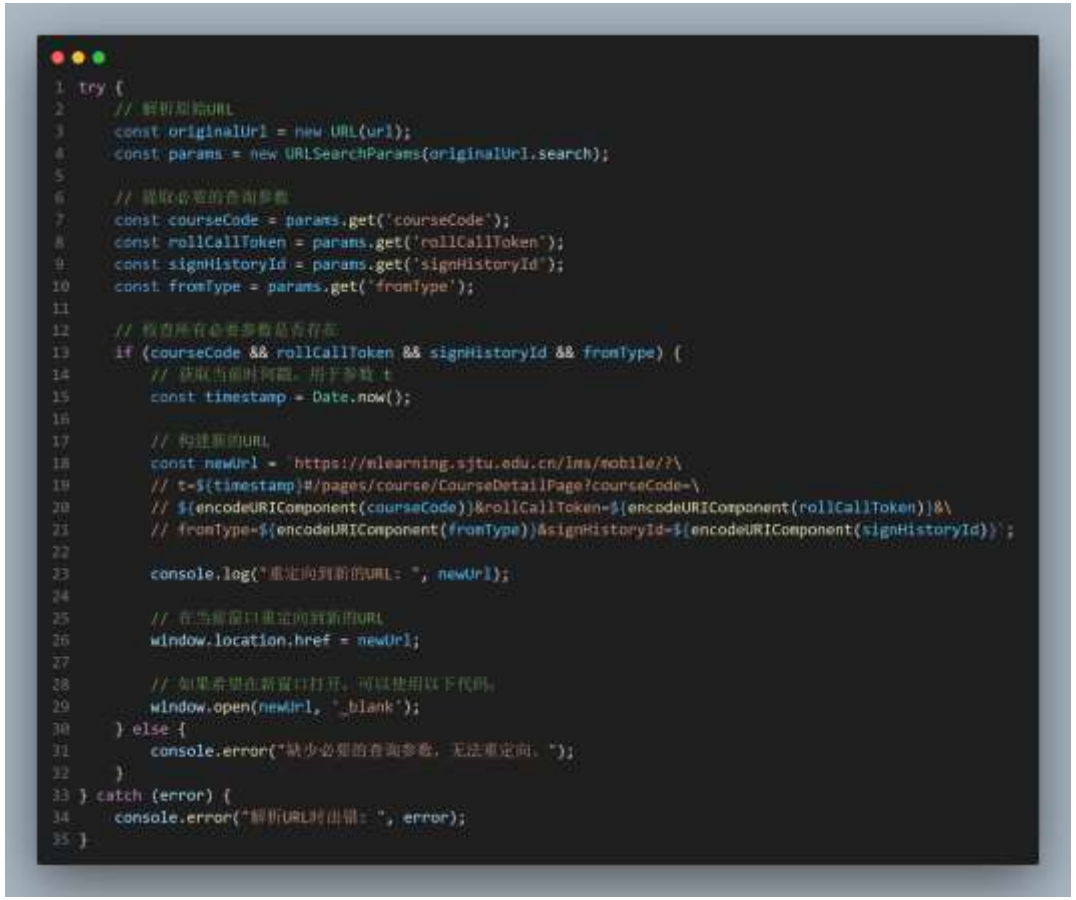
https://mlearning.sjtu.edu.cn/lms/mobile/forscan/?courseCode=70185&rollCallToken=dacc8158b7ac4d468166ca3905b4d85c_NORMAL_97eadb527ab5b4ae1c1e46bb43cabe86&signHistoryId=dacc8158b7ac4d468166ca3905b4d85c&fromType=scanRollCall

https://mlearning.sjtu.edu.cn/lms/mobile/?t=1618827503000/#/pages/course/CourseDetailPage?courseCode=70185&rollCallToken=dacc8158b7ac4d468166ca3905b4d85c_NORMAL_97eadb527ab5b4ae1c1e46bb43cabe86&fromType=scanRollCall&signHistoryId=dacc8158b7ac4d468166ca3905b4d85c

二维码动态参数探因

- 调试的时候发现
- 如果直接访问给定链接，会出现404。
- 通过不同终端（手机或者Edge侧边栏）打开和浏览器实际访问的链接来对比，发现要正确访问是需要带有t这个参数信息来校验的。
- 通过同一课程的不同时间段id来检测，发现t是一样的，所以这个t应该是学生的某一种id，再通过其它课程的签到数据，发现这个t对于同一个学生是固定。但是发现在浏览器中删掉t参数也能正常访问——只要有提前登录过，浏览器存下了cookies。所以，要自动签到，只需要格式上符合后者的格式即可。
- 这里我们可以顺便复习下URL的规则，这里也涉及SPA网页的内容。
- 而通过同一个签到事件的不同签到码来看，上述rollCallToken后续的标红部分也是动态生成的，刷新时间对应着这个token的持续时间。

二维码动态参数



```
try {
    // 解析原始URL
    const originalUrl = new URL(url);
    const params = new URLSearchParams(originalUrl.search);

    // 提取必要的查询参数
    const courseCode = params.get('courseCode');
    const rollCallToken = params.get('rollCallToken');
    const signHistoryId = params.get('signHistoryId');
    const fromType = params.get('fromType');

    // 检查所有必要参数是否存在
    if (courseCode && rollCallToken && signHistoryId && fromType) {
        // 获取当前时间戳，用于参数 t
        const timestamp = Date.now();

        // 构建新的URL
        const newUrl = `https://mlearning.sjtu.edu.cn/lms/mobile/?\
        // t=${timestamp}#/pages/course/CourseDetailPage?courseCode=\
        // ${encodeURIComponent(courseCode)}&rollCallToken=${encodeURIComponent(rollCallToken)}&\
        // fromType=${encodeURIComponent(fromType)}&signHistoryId=${encodeURIComponent(signHistoryId)}`;

        console.log("重定向到新的URL：", newUrl);

        // 在当前窗口重定向到新的URL
        window.location.href = newUrl;

        // 如果希望在新窗口打开，可以使用以下代码：
        window.open(newUrl, '_blank');
    } else {
        console.error("缺少必要的查询参数，无法重定向。");
    }
} catch (error) {
    console.error("解析URL时出错：", error);
}
```

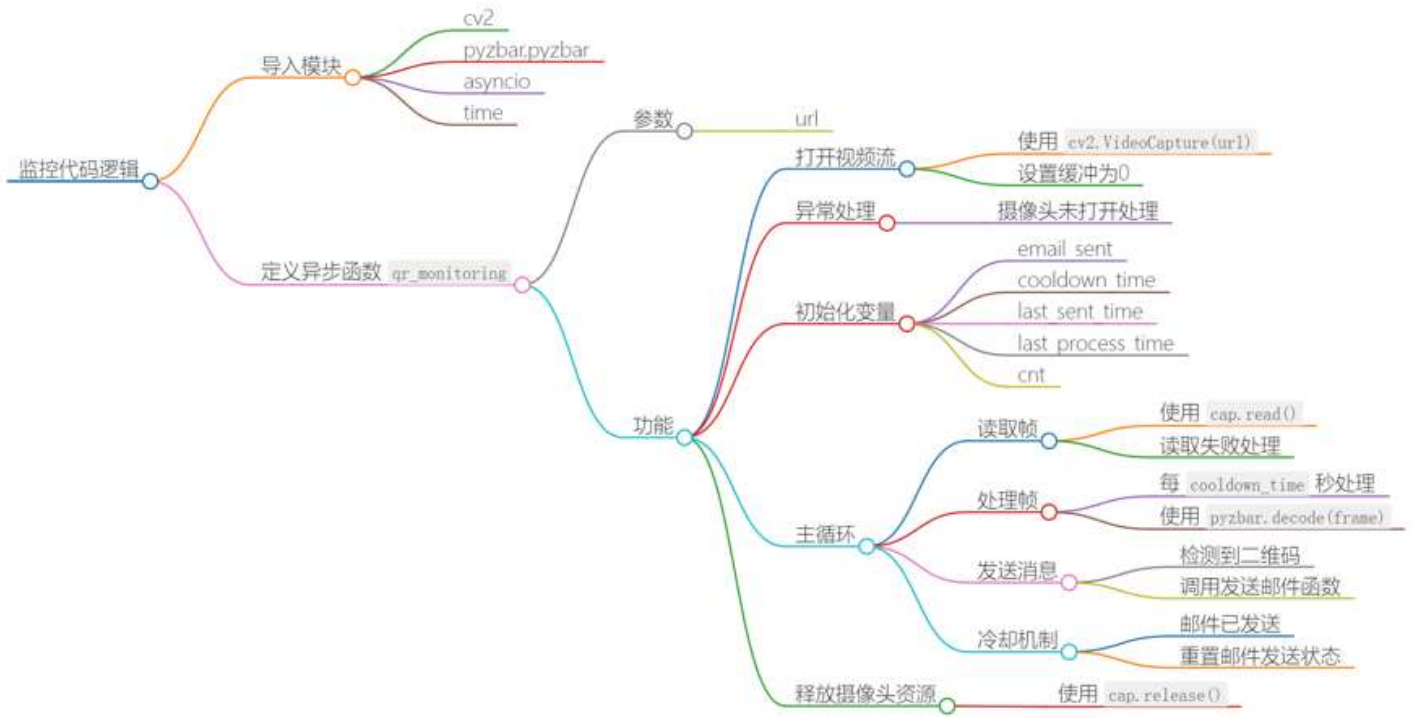
- 解析参数并且转化成正确格式

- 这里的t随机生成，实验发现随便填也能正确访问
  - Jaccount的机制复杂，所以我们也用采用浏览器插件的形式，利用好浏览器自己存储的cookies。

后端代码逻辑

- FastAPI搭建API服务
- Send_email
- Is_valid_url
- Is_wanted_url
- Qr_monitoring
- @app.websocket(“/ws”)

监控程序基本逻辑


监控代码逻辑
导入模块
cv2
pyzbar.pyzbar
asyncio
time
定义异步函数 qr_monitoring
参数
url
功能
打开视频流
使用 cv2.VideoCapture(url)
设置缓冲为0
异常处理
摄像头未打开处理
初始化变量
email_sent
cooldown_time
last_sent_time
last_process_time
cnt
主循环
读取帧
使用 cap.read()
读取失败处理
处理帧
每 cooldown_time 秒处理
使用 pyzbar.decode(frame)
发送消息
检测到二维码
调用发送邮件函数
冷却机制
邮件已发送
重置邮件发送状态
释放摄像头资源
使用 cap.release()

获取最新帧

CV库和cv.read()的简单理解

```
1 async def qr_monit
2     cap = cv2.Vide
3     # 怎么确保实时性？设置缓冲为0
4     cap.set(cv2.CAP_PROP_BUFFERSIZE, 0)
```

发送邮件

简单但是实用

share一下，以后可以和一些服务器任务监控结合起来。比如跑一个两三天的程序，把这个程序的结束作为trigger

一个监控进程，某个trigger触发后，调用send_email

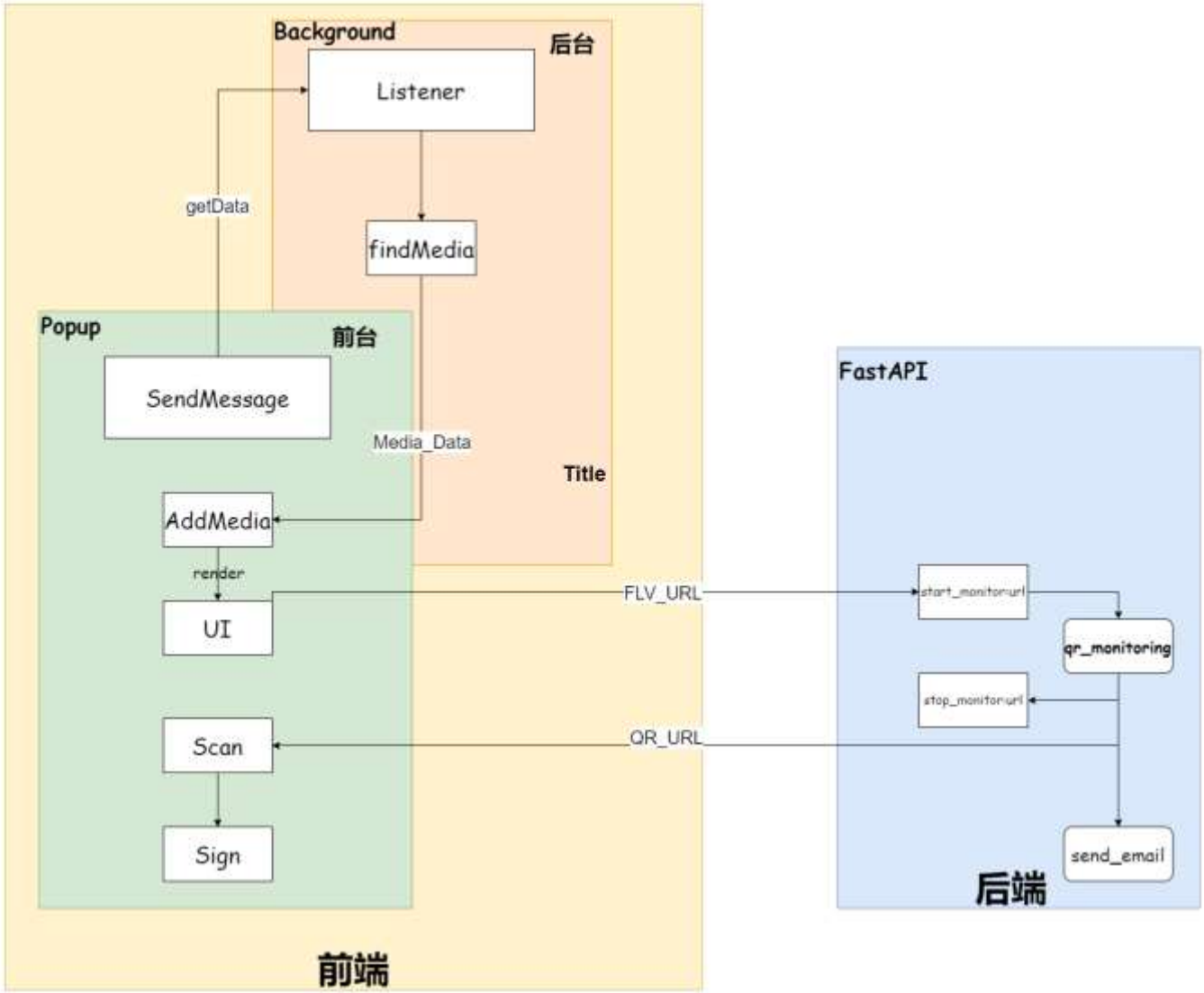
```python
import smtplib
# 邮件发送函数
def send_email(subject, email_body, to_email=NOTIFY_EMAIL):
    # 创建邮件
    msg = MIMEMultipart()
    msg['From'] = SMTP_USERNAME
    msg['To'] = to_email
    msg['Subject'] = subject

    # 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)
        #以QQ邮箱为例，SMTP_PASSWORD是开通SMTP服务时生成的授权码
        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}")
```

更多功能与QA

更多功能……

- 性能优化
  - 多个websocket
  - 并发执行
  - 从C/S模式更改为P2P的模式，把后端部分部署为迅雷一样的应用。

- 功能扩展（不局限于扫码）
  - 检测diff，生成摘要
  - 检测高光时刻（借助模型），生成切片指导


Background
后台
Listener
getData
findMedia
Popup
前台
SendMessage
Media_Data
Title
AddMedia
render
UI
Scan
Sign
前端
FLV_URL
QR_URL
FastAPI
start_monitor:url
qr_monitoring
stop_monitor:url
send_email
后端

QA