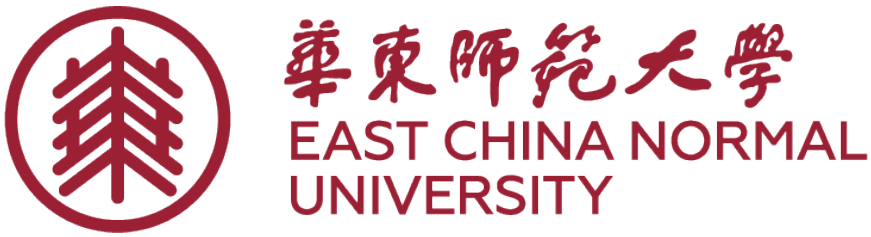


ECNU 校园插件项目报告

ECNU Campus Plugins - Project Report

关卓谦 张梓卫 王文锦

指导老师：陈良育

2025 年 1 月 6 日

华东师范大学软件工程专业

目录

一 项目概述 **3**
1 项目背景 . . . 3
2 项目信息 . . . 3
3 项目功能 . . . 3

二 项目整体架构 **3**

三 项目模块详述 **4**
1 插件登录 ECNU 及辅助登录 . . . 4
1 .1 辅助登录大致实现 . . . 4
1 .2 无状态登录原理 . . . 5
2 插件框架模块 . . . 5
2 .1 插件加载器（PluginLoader） . . . 5
2 .2 插件事件方法 . . . 6
2 .3 插件配置（PluginConfig） . . . 6
2 .4 插件缓存（PluginCache） . . . 7
2 .5 插件上下文（PluginContext） . . . 7
2 .6 插件消息传递（Message） . . . 8
3 课前提醒模块 . . . 8
3 .1 模块设计与配置 . . . 8
3 .2 课表获取及数据处理原理 . . . 9
4 研修间预约模块 . . . 10
4 .1 代码框架及配置 . . . 10
4 .2 研修间预约实现原理 . . . 10
5 图书馆预约模块 . . . 12
5 .1 代码框架及分析 . . . 12
5 .2 图书馆预约实现原理 . . . 13
6 电费查询模块 . . . 15
6 .1 插件代码结构 . . . 15
6 .2 插件配置页面 . . . 16
6 .3 电费查询原理 . . . 16

四 项目主要界面贴图 **17**
1 主页 . . . 17
2 插件配置页面 . . . 17
3 系统托盘图标 . . . 18

五 总结 **18**
1 项目前景 . . . 18
2 项目收获 . . . 19

一 项目概述

1 项目背景

本项目旨在为 ECNU 校园提供支持，专注于解决校园生活中的痛点，并通过自动化功能提升便利性与体验。

2 项目信息

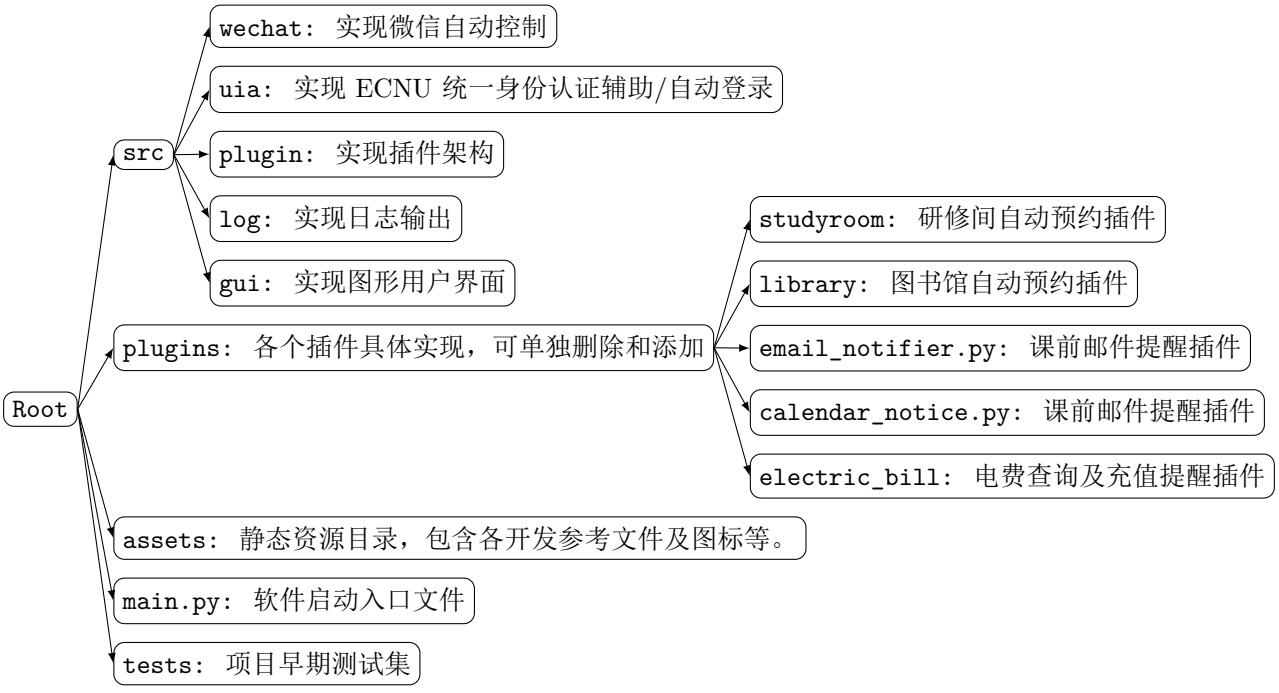
- 项目作者：关卓谦（10235101529）、张梓卫（10235101526）、王文锦（10235101510）
- 项目仓库（含部署方法）：https://github.com/azazo1/ecnu-campus-plugins
- 项目基于 Python 3.12 开发，基于 Pyside 6 实现图形界面，课程报告使用 LaTeX 编写。
- 本项目遵循 MIT 开源协议，供感兴趣者学习与交流。其中，LaTeX 宏包部分使用 LPPL 授权。
- 华东师范大学校徽图案版权归华东师范大学所有。

3 项目功能

本项目包含：

- 一键生成当周课表、课前邮件提醒
- 课后研修间与图书馆的全自动预约
- 宿舍电费自动查询及充值提醒
- 适配 ECNU 多种系统的登录缓存，实现了插件框架，能够接收更多开发者的贡献与集成

二 项目整体架构



三 项目模块详述

1 插件登录 ECNU 及辅助登录

插件许多功能需要调用 ECNU 提供的各种接口。接口的调用依赖于 ECNU 各个系统的登录缓存（在项目内部保存为 `LoginCache`）。为了获取有效的登录缓存简化登录操作，插件登录使用了 WebDriver 控制浏览器实现自动化或半自动化的辅助登录流程。

> 本部分代码位于 `/src/uia/login.py` 文件中，核心函数 `get_login_cache` 已提供了完整的注释信息。

在插件登录时，会出现浏览器 ECNU 统一身份认证的界面，登录有两种方式。

- 方法一：填写图 1 中的表单实现登录。
- 方法二：扫描图 2 中的二维码实现登录。

图 1: ECNU UIA 登录界面（表单）

图 2: ECNU UIA 登录界面（二维码）

提示：切换表单登录和二维码登录

表单登录：可以在项目根目录创建文件 `login_info.toml` 然后填写如下内容（替换尖括号以内的部分）。插件登录时会读取此文件中的学号密码，自动填写图 1 中的表单，实现全自动辅助登录。

```
stu_number = "<填写学号>"
password = "<填写数据库密码>"
```

二维码登录：删除 `login_info.toml` 文件，此时插件辅助登录时会截取二维码图片并发送至邮箱提醒。可以使用微信扫描二维码或用微信打开邮件中的连接实现半自动登录。

1 .1 辅助登录大致实现

当此软件执行辅助登录的时候：

二维码登录 进入 ECNU 统一身份认证 (下面简称 UIA) 界面之后，辅助登录默认使用二维码登录而不是表单登录。二维码登录时，脚本截取 UIA 二维码登录图片然后发送到目标邮箱，通过用户微信扫描二维码或者微信打开邮箱中的连接来实现登录。

表单登录 如果创建了 `login_info.toml` 文件（步骤见上述提示），那么脚本会读取其中的学号和密码并自动填写到 UIA 表单中。表单中的验证码通过开源库 DDDDocr 来识别，成功识别四位验证码之后模拟点击登录按钮来登录。如果验证码识别失败则刷新重试。

成功登录 UIA 之后，各个插件通过自己实现的缓存抓取函数（*Cache Grabber*）在各个 ECNU 系统中获得所需要的登录缓存。最终，登录缓存通过插件加载器（*Plugin Loader*）分发给各个插件。

1 .2 无状态登录原理

我们获取的 `LoginCache` 置于项目根目录中的文件 `login-cache.pickle` 中，其中最重要的字段为 **Bearer** 字段以及 **cookie** 字段，这是在 **OAuth 2.0** 框架下在 HTTP 请求头中的认证类型，携带访问令牌通过 **Bearer Token** 来实现无状态的认证。

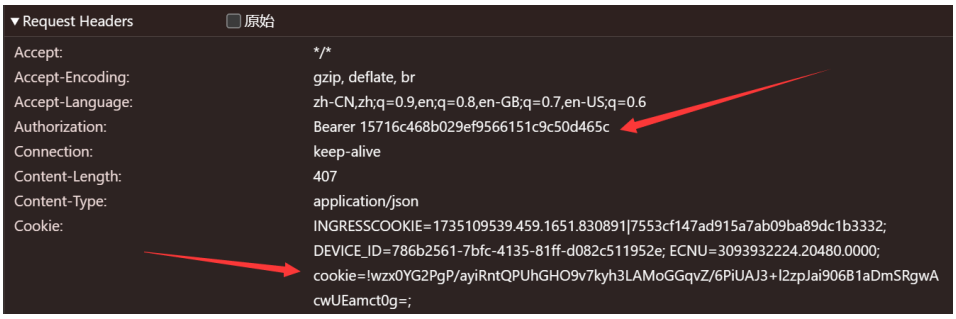


图 3: Main Cookie 及鉴权 Token

正因该 Bearer 字段存在一次获取后便保持一定时长活性的特性，我们才能实现登录缓存的保存机制，为后续分发各缓存至插件提供逻辑闭环（见章节 2 .1）。有时 OCR 的结果不是四位有效的验证码，所以我们添加了一行代码增大了准确性：

```
if len(captcha) != 4:
    driver.refresh()
```

尝试使用后，它有将近 $\frac{2}{3}$ 识别的准确率。此后，便可以实现自动化登录，解放双手了。较为幸运的是，华师并未设置验证码多次登录失败后禁止登录的逻辑，所以成功率较低的情况下也可以照常使用。

2 插件框架模块

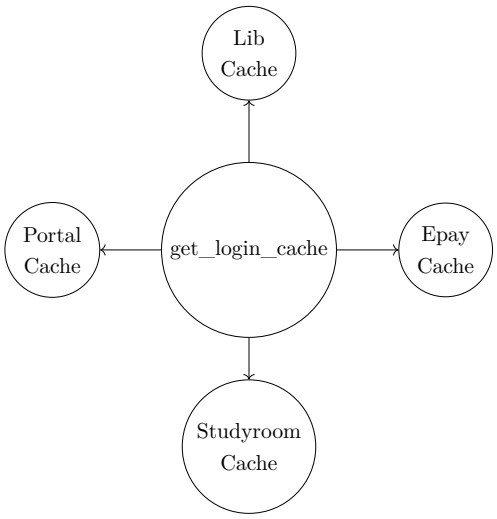
为了合理地分离代码中的各个逻辑部分，项目构建了插件框架，将不同的功能划分到一个个插件中。插件框架实现了单独开启和关闭每个插件的功能，让使用者可以根据需要选择希望使用的功能。

2 .1 插件加载器（PluginLoader）

插件加载器（*PluginLoader*）类在 `/src/plugin/__init__.py` 中定义。

项目启动时，PluginLoader 依次导入位于项目根目录下的 `plugins` 文件夹中的插件。然后从文件中读取各个插件配置，并传递给插件。插件配置读取完毕之后，PluginLoader 加载各个插件。被加载的插件可以从其他插件中接收消息和按照特定周期执行。

插件的加载和运行通过 PluginLoader 驱动。它可调用 `get_login_cache()`，实现各缓存的分发，供相应的插件调用。



PluginLoader 具有缓存分发功能：

- **Lib Cache：** 用于图书馆管理系统。
- **Studyroom Cache：** 用于研修间自动预约系统。
- **Portal Cache：** 用于公共数据库页面的缓存，目前仅用于课表的获取。
- **Epay Cache：** 用于华东师范大学校园卡管理页面的自动查询电费功能。
- **More Cache...：** 适配的框架可以获取其余 ECNU 页面的 Cache 以供开发者使用。

表 1: PluginLoader 的缓存分发与功能说明

2 .2 插件事件方法

插件注册时，应继承自 Plugin 基类，并自定义实现插件的各种事件方法。
其中，插件（*Plugin*）基类在 `/src/plugin/__init__.py` 中定义。

成功注册后，插件的各种事件方法会被触发，主要的事件方法为：

插件加载和停止（Load 与 Unload）事件 当插件被加载和停止时触发，插件的加载和停止在插件被导入之后，可能会被使用者多次触发。用于标记当前插件是否正在运行，方便插件释放资源。代码示例如下：

```
@register_plugin(
    name="demo_plugin"
)
class DemoPlugin(Plugin):
    def on_load(self, ctx: PluginContext):
        pass
    def on_unload(self, ctx: PluginContext):
        pass
```

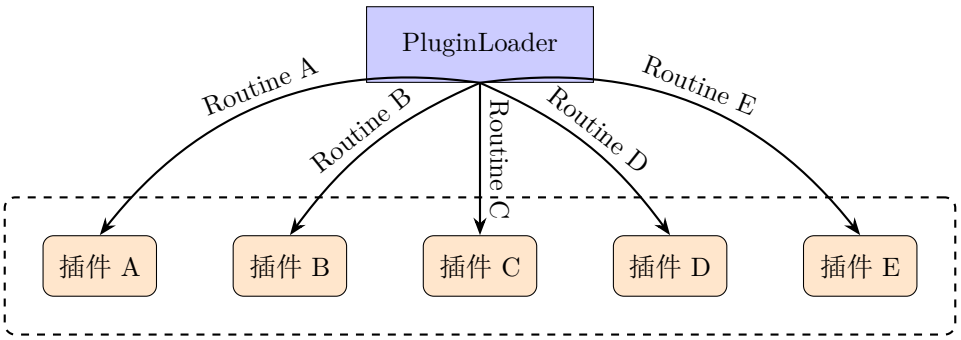
插件加载和停止示例

插件周期事件（Routine） 插件在注册的时候可以提供一个回调周期，插件会在指定的回调周期得到回调。示例如下：

```
@register_plugin(
    name="demo_plugin",
    routine=Routine.MINUTELY
)
class DemoPlugin(Plugin):
    def on_routine(self, ctx: PluginContext):
        pass # 周期性得到回调
```

周期事件示例

> **Note**
>
> **PluginLoader** 根据预定义的频率（Secondly、Minutely、Hourly、Daily、Weekly），调用每个插件的 `on_routine` 方法。插件会根据任务频率接收并处理任务，完成对应的功能逻辑。
>
> 箭头上的 Routine 统一表示调度逻辑，具体任务频率的细节由配置文件决定。例如：



2 .3 插件配置（PluginConfig）

此部分代码如 `/src/plugin/config.py` 所示。

插件在插件注册时提供需要的配置，在 PluginLoader 从文件中读取插件配置的时候，获得配置的值。

插件配置会集中保存在 `plugin_config.toml` 中，GUI 会为每个插件配置自动生成交互式修改组件，见 2 。使用者可以通过修改文件或者通过 GUI 修改插件配置。示例如下：

```python
@register_plugin(
    name="demo_plugin",
    configuration=PluginConfig()
    .add(TextItem(name="first_name", default_value="Tom", description="Your first name"))
)
class DemoPlugin(Plugin):
    def on_config_load(self, ctx: PluginContext, cfg: PluginConfig):
        first_name = cfg.get_itme("first_name").current_value
```

插件配置示例

> **Note**
>
> ItemType 中我们定义了许多不同的配置项类型，例如 TextItem、NumberItem、DateItem、Password 等。通过不同类型的定义，可以接受不同的输入，防止用户输入的内容不符合要求，而像 Password 的定义处理，可以在插件配置界面不明文展示密钥信息，充分保障了信息的安全性。

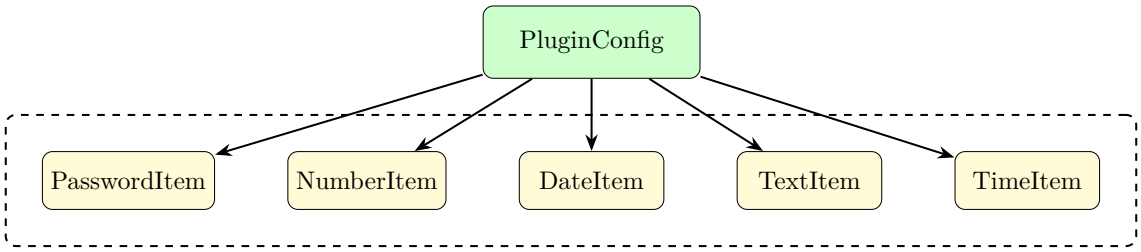


图 4: PluginConfig 和所有 ConfigItem 的关系示意图

2 .4 插件缓存（PluginCache）

插件内部生成的数据可以得到统一的持久化保存。插件只需对 PluginCache 进行修改，数据会在插件停止时保存，在插件加载时再次读取。

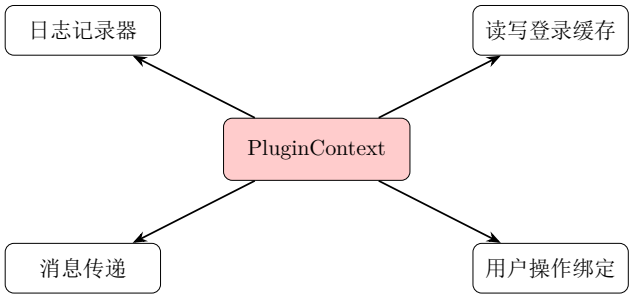
```python
@register_plugin(
    name="demo_plugin"
)
class DemoPlugin(Plugin):
    def on_load(self, ctx: PluginContext):
        ctx.get_logger().info(ctx.get_cache().get("time"))
    def on_routine(self, ctx: PluginContext):
        ctx.get_cache().set("time", time.time())
```

插件缓存示例

2 .5 插件上下文（PluginContext）

此部分代码见 `/src/plugin/context.py`

> **Note**
>
> 为了保证项目的整洁和规范性，插件创建文件和记录日志等操作使用 PluginContext 进行上下文处理。
> 每个插件均拥有一个上下文环境，使插件能够与框架进行交互和访问必要的资源。



PluginContext 的功能组成：

- **日志记录：**插件专属的日志记录（信息、警告、错误等）。
- **读写登录缓存：**插件可以通过 `ctx.get_cache()` 访问自己的持久化缓存，用于存储跨会话的数据。缓存支持数据的读取和写入，确保插件状态的持久性。
- **消息传递：**向其他插件发送消息
- **用户操作绑定：**可创建可交互按钮（Bind Action），相应的回调函数执行对应操作。

2 .6 插件消息传递（Message）

插件通过 PluginContext 可以向其他插件发送消息。发送示例如下：

```python
@register_plugin(
    name="send_demo_plugin"
)
class DemoPlugin(Plugin):
    def on_routine(self, ctx: PluginContext):
        ctx.send_message("recv_demo_plugin", "Hello World")

@register_plugin(
    name="recv_demo_plugin"
)
class RecvPlugin(Plugin):
    def on_recv(self, ctx: PluginContext, from_plugin: str, obj: Any):
        ctx.get_logger().info((from_plugin, obj))
```

发送消息示例

3 课前提醒模块

3 .1 模块设计与配置

本模块位于 `/plugins/studyroom/calendar_notice_plugin.py`

本插件在设计上，主要用于与其他插件通信。查询最近的课表并发送邮件提醒用户只是最简单的功能。

> **Note**
>
> 通过插件通信，可以通知其他插件（如研修间预约与图书馆预约模块），若一定时间内没有课程，其他模块便可以开始调用相应的预约接口。我们的设计为：查询当前时刻至次日该时刻用户的课程安排，如果有课程，上课前的一定时段发送邮件提醒用户去上课。可以通过插件配置来配置上课前多久提醒用户。

图 5: 课前提醒模块配置

3 .2 课表获取及数据处理原理

图 6: ECNU Portal 课表页面

进入电脑端 Portal 主页面右侧，我们可以看到自己的课表，使用 Selenium-Wire 抓包得知，实际上，本课表是存在一个请求 Url 的。

那么我们通过上述获得的 `login_cache`，携带 **Bearer** 鉴权字段和对应的 **Cookie**，通过 requests 库发起请求，即可实时获取课表。

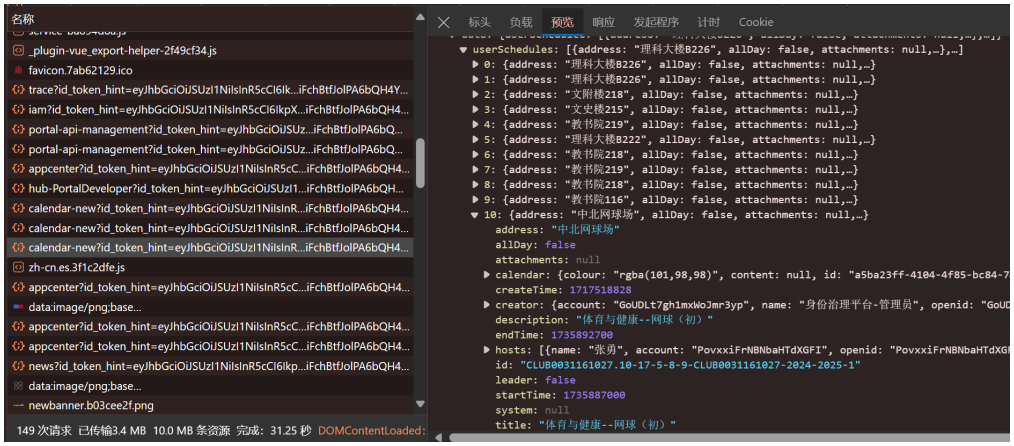


图 7: Portal 课表请求

这个 POST 请求采用的是 GraphQL 的查询格式，我们只需要使用 filter 过滤器查询自己所需要的字段即可。

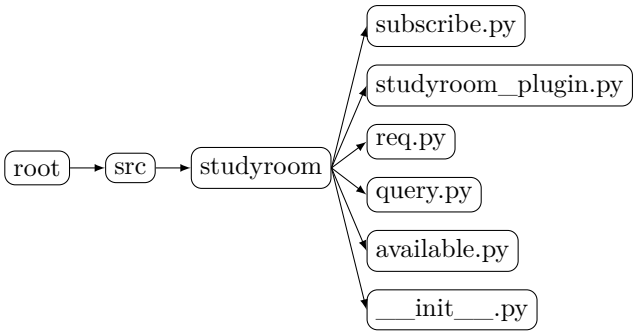
```
USER_SCHEDULES = """
query ($filter: ScheduleFilter, $userId: String) {
  userSchedules(filter: $filter, userId: $userId) {
    address # 上课地点
    hosts {
      name # 教师名字
    }
    description # 课程信息和描述
    endTime # 结束时间
    startTime # 开始时间
  }
}
"""
```

- 获取用户从此刻至第二天此时的课表后，将每一个课程的时间戳转换后与当前的时间进行比较，
- 若小于了用户设置的提醒时间，则发送邮件提醒用户。

4 研修间预约模块

4 .1 代码框架及配置

此部分代码位于 `/src/studyroom/` 中。



模块说明：

Query (`query.py`)
该模块仅实现对 `url` 的简单请求，不作任何数据处理。它通过相应的 API 获取研修室的可用性和详细信息。

Available (`available.py`)
`available.py` 模块负责处理房间可用性数据。它分析当前和已预订的时间段，确定研修室可预约的时间段。

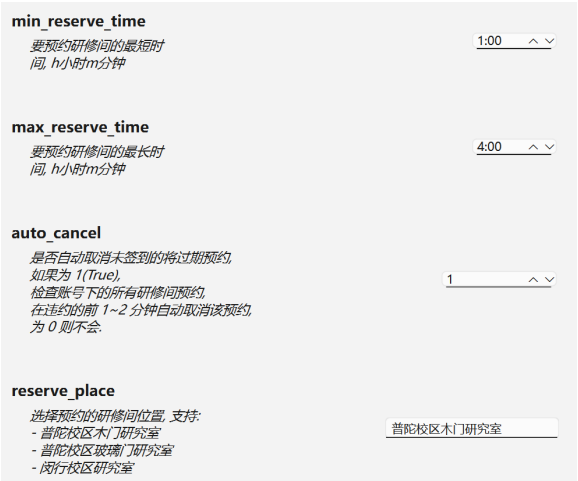
Subscribe (`subscribe.py`)
`subscribe.py` 模块管理预约流程。它包括进行预约、检查预约状态以及在特定条件下自动取消预约的功能。

图 8: 研修间的目录结构及模块说明

配置好后，软件会自动计算当前时间和下次上课时间的间隔，并判断是否符合预约条件。

研修间配置说明：

- **min_reserve_time：** 最短预约时长。最小值为 `1:00`。
- **max_reserve_time：** 最长预约时长。最大值为 `4:00`。
- **auto_cancel：**
  - `1` (**True**)：自动取消即将过期的预约。
  - `0` (**False**)：不自动取消。
- **reserve_place：**
  - 普陀校区木门研修室、玻璃门研修室
  - 闵行校区研修室



4 .2 研修间预约实现原理

华东师范大学研修间预约最少预约时长在 60 分钟以上，最长不能超过 240 分钟。对于单人学生，仅有普陀校区木门研究室、闵行校区研究室、普陀校区玻璃门研究室可供单人预约使用。

查看研修间的预约情况，我们首先从这一张研修间状态列表中计算出可用时间（AvailableInfos）都有哪些：

图 9: 研修间状态示例

加载入网页时，该 Url 会自动调用：https://studyroom.ecnu.edu.cn/ic-web/roomDevice/roomAvailable 拥有查询当前类别的研修间的功能，其返回的字段如下：

```
{'devId': 3676503,                                  # 设备 ID
'devName': '普陀校区单人间C421',                     # 设备名称
'minResvTime': 60,                                  # 最小预约时间
'openTimes': [{'openEndTime': '22:00',              # 开放结束时间
               'openLimit': 1,                      # 最少预约人数
               'openStartTime': '08:00'}],          # 开放开始时间
'resvInfos': [{'resvBeginTime': '2024-12-26 '       # (People No.1) 的预约信息
                                '17:01:00',
               'resvEndTime': '2024-12-26 '
                              '21:01:00',
               'resvStatus': 1093}]},
```

它只为我们提供了 'resvInfos' 字段，这应该是用于前端供渲染黄色已预定时间段的给用户的，所以我们需要将它与 'openTimes' 字段结合起来，得到研修室的可用时间段。

Note

- 将 [openStartTime, currentTime] 区间视为不可用时间段。例如当天 18:55 P.M 前的时间段都无效。
- 仅 AvailableTime > minResvTime 时，才将这段区间设置为 AvailableTime, 不考虑未超过最短预约时间的情况。

返回的信息字典中，我们通过程序包含了一个新字段 availableInfos，这样就可以提供给用户可用的时间段了。

之后，使用 reserve.py 中的函数 _fetch_userInfo 来获取 appAccNo，这是识别用户身份的字段，在发送 reserve 请求时，需要传入该字段作为 Payload 的一部分。最后通过调用 reserve_room 函数即可完成全自动预约。

Note

每一次预约都会形成一个唯一的预约编号，称为 uuid，我们在进入个人中心时，网页会自动调用查询的接口：https://studyroom.ecnu.edu.cn/#/ic/userinfo ，所以，我们也可以通过抓包获取 uuid，便可以知道用户当前是否有预约了，这为后续的自动取消预约提供了保障。

研修间预约时，需要提交的表单样式如下，我们不采用浏览器自动操作的形式，而是截取提交时发送的 url 请求：

图 10: 研修间预约表单

抓包得到 POST 请求的 Url 如下：https://studyroom.ecnu.edu.cn/ic-web/reserve
只需传入相关的字段即可：

图 11: 研修间预约请求

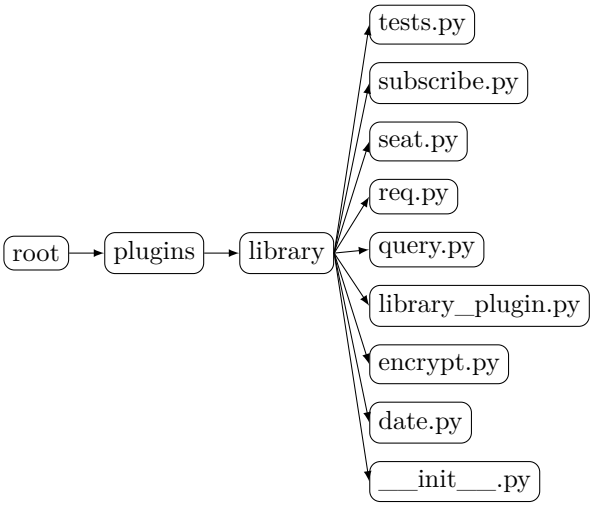
```python
headers = {
    "Cookie": f"ic-cookie={ic_cookie}",
} # 从 StudyroomCache 中获取 ic-cookie

# 从 _fetch_userInfo 获取用户 ID
appAccNo = self._fetch_userInfo().get("accNo")

payload = {
    "sysKind": 1,  # 系统类型，默认为 1
    "appAccNo": appAccNo,
    "memberKind": 1,  # 成员类型，默认为 1
    "resvBeginTime": resvBeginTime,
    "resvEndTime": resvEndTime,
    "testName": testName,
    "resvMember": [appAccNo],
    # 默认预约人员列表只有当前用户
    "resvDev": resvDev,
    "memo": memo,
}
```

Listing 1: 预约请求需要传入的字段

5 图书馆预约模块

本模块位于 /src/plugins/library 中。

5 .1 代码框架及分析



部分模块功能说明：

Encrypt (encrypt.py)
提供加密与解密功能，使用 AES 加密算法保障数据传输安全，通过填充和密钥生成实现加密。

Seat (seat.py)
提供座位数据对象化表示，实现计算座位间距离、判断空闲状态等功能，并包括寻找最优座位的算法。

Subscribe (subscribe.py)
实现座位预约的核心逻辑，包括预约、查询、取消功能，并通过加密模块保障传输安全。

Library Plugin (library_plugin.py)
提供座位预约插件，支持快速预约、用户配置偏好和自动取消未签到预约等功能。

Query (query.py)
封装查询接口，包括查询座位空闲情况、具体座位信息、可用时间段等，并支持按条件筛选区域和座位。

图 12: ECNU 图书馆预约模块说明

配置说明：

- **prefer_study_duration**：偏好的学习时长
- **auto_cancel**：是否自动取消未签到的过期预约
  - 1 (True)：会在预约到期前 1～2 分钟自动取消
  - 0 (False)：不会自动取消
- **premise**：预约座位选择的校区

表 2: 图书馆预约配置

5 .2 图书馆预约实现原理

系统使用了 Axios 的请求拦截器来对特定 API 请求进行加密。以下是相关的 JavaScript 代码片段：

```
service.interceptors.request.use(function(config) {
    let encryptRequest = ["/api/login/resetpass", "/api/login/login", "/api/login/forget", "/api/Seat/confirm",
    "/api/Seminar/confirm", "/reserve/index/confirm", "/api/Enter/confirm", "/api/Seat/touch_qr_books", "/api/
    seat/qrcode", "/api/seat/qrcode_not_card", "/api/Study/StudyOrder", "/api/login/updateUserInfo", "/api/seat/
    xuzuoconfirm"],
    token = sessionStorage.getItem("token") || "",
    lang = localStorage.getItem("lang") || "zh";

    if (token) {
        // 处理已登录状态下的请求
        if (encryptRequest.includes(config.url)) {
            let encryptedData = encrypt("encrypt", config.data);
            config.data = encryptedData;
        }
        // 其他处理逻辑...
    } else {
        // 处理未登录状态下的请求
        if (encryptRequest.includes(config.url)) {
            let encryptedData = encrypt("encrypt", config.data);
            config.data = encryptedData;
        }
        // 其他处理逻辑...
    }
    // 其他配置...
    return config;
});
```

Listing 2: 请求拦截与加密代码

请求拦截器解析 上述代码通过 Axios 的请求拦截器，对特定 API 路径的请求数据进行加密处理。主要步骤包括：

1. 定义需要加密的 API 路径数组 `encryptRequest`。
2. 从 `sessionStorage` 和 `localStorage` 中获取 `token` 和 `lang`。
3. 判断当前请求的 URL 是否在需要加密的列表中。
4. 如果需要加密，则调用 `encrypt` 函数对请求数据进行加密。

加密函数分析 加密操作主要集中在 `encrypt` 函数中。以下是该函数的具体实现：

```
    function encrypt(g, y) {
        var k = exchangeDateTime(new Date, 41);
        // `.split("")`: 按照每个字符分割字符串.
```

```
        k = `${k}${k.split("").reverse().join("")}`; // AES 密钥.
        // 调试发现 k 可能为 "2024112882114202", 就是当前日期年月日这个字符串和他的倒转相加.
        var j = k;
        var pe = "ZZWBKJ_ZHIHUAWEI"; // AES iv 向量.
        if (g == "encrypt")
            // 加密操作, 见下文.
            return crypto.encrypt(JSON.stringify(y), j, pe);
        ...
    }
    const CryptoJS = cryptoJs.exports
    const crypto = {
        encrypt(g, j, pe) { // 这里的 j 就是密钥, 就是上文 encrypt 中的 k 变量.
            // CryptoJS.enc.Utf8.parse 是 CryptoJS 中的一个方法
            // 用于将普通字符串转换为 CryptoJS 的 WordArray 对象, 不涉及加密操作.
            var j = CryptoJS.enc.Utf8.parse(j);
            var pe = CryptoJS.enc.Utf8.parse(pe);
            var Ce = CryptoJS.AES.encrypt(g, j, { // CryptoJS.AES.encrypt(message, key, options)
                iv: pe,
                mode: CryptoJS.mode.CBC,
                padding: CryptoJS.pad.Pkcs7
            });
            return Ce.toString()
        }
    };
```

Listing 3: 加密函数代码

加密过程详解

1. **密钥生成：**
   - 调用 `exchangeDateTime` 函数获取当前日期时间，并传入参数 41 进行处理，生成基础密钥 `k`。
   - 将 `k` 与其倒序字符串拼接，形成最终的 AES 密钥。
   - 例如，如果 `k` 为"2024112882114202"，则最终密钥为"2024112882114202202411288211420"。
2. **初始化向量（IV）：**
   - 固定 IV 值为"ZZWBKJ_ZHIHUAWEI"。
3. **加密操作：**
   - 使用 `CryptoJS` 库进行 AES 加密，采用 CBC 模式和 Pkcs7 填充。
   - 将待加密的数据序列化为 JSON 字符串后进行加密。
   - 返回加密后的 Base64 字符串。

Python 实现加密过程

```
    def encrypt(cls, json_data: dict, key: str = None) -> str:
        """
        加密函数, 加密数据并返回加密 base64 字符串。

        >>> Encryptor.encrypt({"seat_id": "3361", "segment": "1508173"}, "2024112882114202")
        '6l1+11NSwbo9Rje1/+pnuSqexfDXg/pPDTK0KJEG/uOIZyucecgEo7VO8ggVRom9'
        """
        if key is None:  # 这个默认值
            key = day_str()
            key = key + key[::-1]
        to_encrypt = json.dumps(json_data, separators=(",", ":")).encode("utf-8")
        en = AES.new(
            key=key.encode("utf-8"),
```

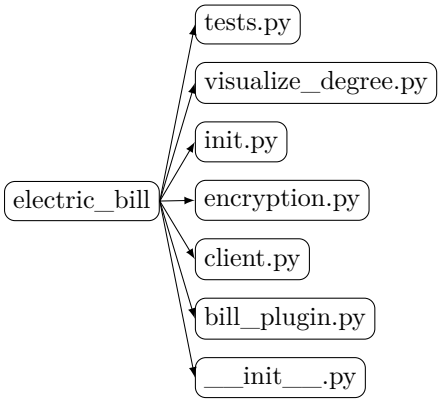
```
            mode=AES.MODE_CBC,
            iv=AES_IV.encode("utf-8"),
        )
        return base64.b64encode(
            en.encrypt(pkcs7_pad(to_encrypt, AES.block_size))
        ).decode('utf-8')
```

Listing 4: Python 加密复现代码

6 电费查询模块

6 .1 插件代码结构

本模块位于 `/src/plugins/electric_bill` 中。



部分模块功能说明：

`bill_plugin.py`
实现电费账单的生成和管理，包含账单计算逻辑、数据处理和存储功能。

`client.py`
负责与服务器的通信，处理客户端请求和响应。包括建立 WebSocket 连接、发送和接收数据，以及处理用户交互。

`encryption.py`
提供数据的加密与解密功能，使用 AES 加密算法保障数据传输的安全性。

`visualize_degree.py`
实现电量使用情况的可视化展示，利用图形库绘制用电趋势图和账单分布图。

图 13: ECNU 宿舍电费查询插件结构树形图与模块功能说明

电费查询插件用到了服务端和客户端，两个客户端之间通过 Websocket 进行通信连接。使用 `iv` 与 `key` 进行加密通信，以确保数据安全与隐私性。本插件采用 `asyncio` 库实现异步编程，确保在进行网络通信、文件操作和浏览器自动化时不会阻塞主线程，提高插件的响应速度和稳定性。

- 客户端主要用于 token 的获取，由于 token 获取需要使用 ECNU 统一登录，故安排在客户端。
- 服务端用于持续对电费账单的查询和记录，并输出电费随时间变化的表格，并将数据提供给客户端。

Note

该插件需要配合 https://github.com/azazo1/ecnu-query-electric-bill 进行使用。
本项目实现的是电费查询的客户端，需要上述仓库的服务端配置。
当用户拥有 ESC 服务器时，可以在 ESC 服务器上进行配置，也可以选择本地部署。

6 .2 插件配置页面

以下是各配置项的简要说明：

- `server_address`: 服务器的 IP 地址或域名，用于客户端与服务器之间的 WebSocket 连接。
- `alert_degree`: 电量警告阈值，当宿舍电量低于此值时，客户端会弹出警告提示用户充值电量。

本插件需要用户手动设置 **key.toml**:

- `key`: 用于 AES 加密的数据密钥，必须为 32 字节长度。
- `iv`: 初始化向量，用于 AES 加密，必须为 16 字节长度。

配置文件的正确设置对于插件的正常运行至关重要。`alert_degree` 可根据实际需求进行调整。
同时，`key.toml` 中的 `key` 和 `iv` 应保持机密，避免泄露以保障数据传输的安全性。

图 14: 插件配置页面示意图

6 .3 电费查询原理

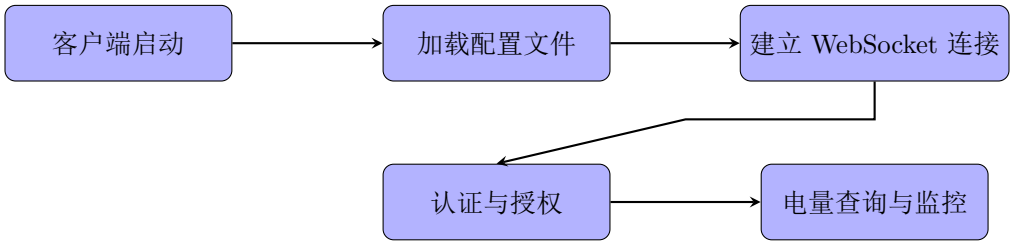
与上述模块的流程相似，先获取 `EpayCache`，之后就可以随时访问 API 获取电费信息。

双端通信 三 .1

- 客户端负责与用户交互、获取电费信息并向服务器发送相关数据；
- 服务器端负责接收客户端的数据请求，处理电费查询，并将结果反馈给客户端。
- 数据在传输过程中采用 AES 加密，确保通信的安全性。

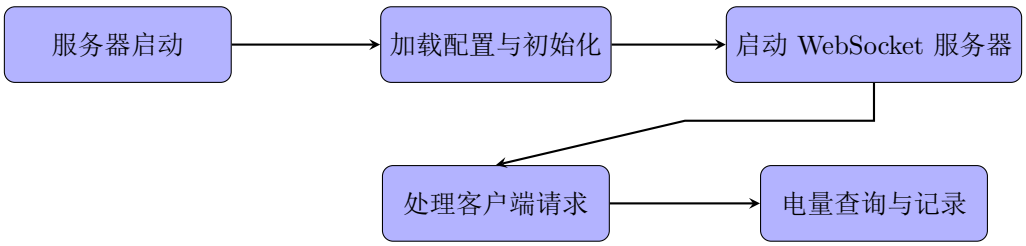
客户端

客户端的启动运行流程如下所示：



服务端

服务端启动的流程如下所示：



服务端可以处理客户端的多种请求：

- 解析客户端发送的加密命令，根据 `type` 字段执行相应操作。
- 支持的命令包括：
  - `POST_TOKEN`：接收并存储客户端的 CSRF Token 和 Cookies。
  - `POST_ROOM`：接收并存储宿舍信息（宿舍号、电区、电楼）。
  - `GET_DEGREE`：返回当前剩余电量。
  - `FETCH_DEGREE_FILE`：返回最近的电量记录文件内容。

四　项目主要界面贴图

整体的 GUI 设计使用 Pyside 6 进行开发，以下是各个页面的贴图及简单介绍。

1　主页

标题栏显示项目标题和登录状态，左侧是标签页切换按钮，右侧是主页大标题和登录、退出软件两个按钮，见图 15

图 15: 主页

2　插件配置页面

插件配置页面右侧框中左侧的列表是插件选择，鼠标单击即可进入各个插件的配置界面，如图16 中红框所示。
在每个插件的配置界面，会根据插件注册时提供的配置项生成可交互配置界面。

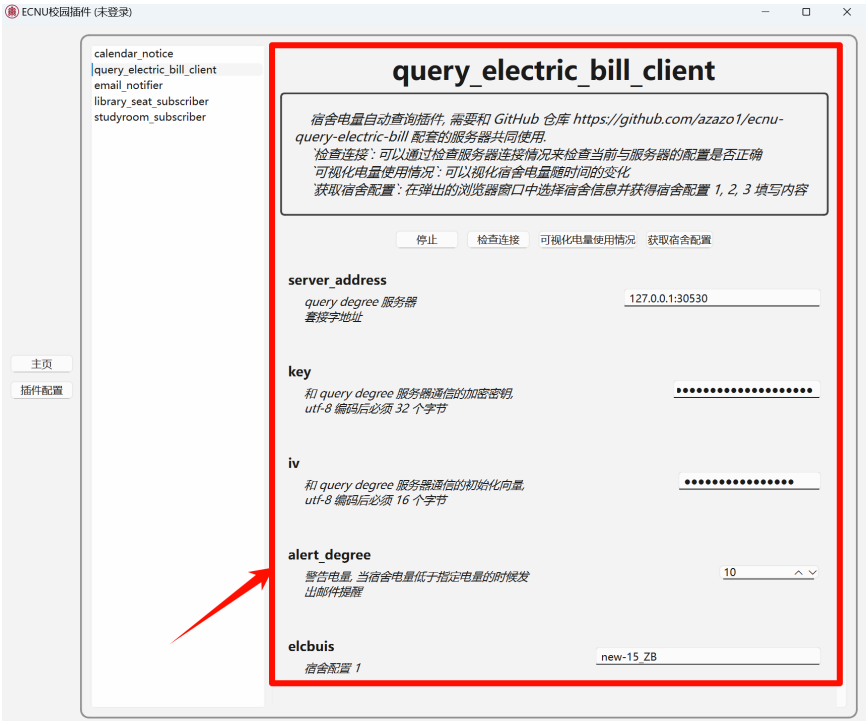


图 16: 插件配置界面示例

3 系统托盘图标

启动项目后，托盘中会出现 ECNU 的图标，鼠标悬停于其上时，可查看当前的登录状态，使用鼠标右键可显示菜单。见图 17a、图 17b、图 17c。

(a) 系统托盘图标

(b) 鼠标悬浮显示状态

(c) 右键显示菜单

五 总结

1 项目前景

由于本项目具备良好的扩展性，后续可以集成更多插件，覆盖更多校园生活场景。例如：

- 支持课表与日历软件（如 Google Calendar）的无缝同步。
- 集成更多校园生活服务，例如自动化导出成绩单、图书馆借阅管理、校园卡消费查询等。

通过持续迭代和优化，本项目有潜力为华东师范大学的师生带来更加便利的校园生活体验。

2 项目收获

关卓谦

Thought 五 .1

在本项目中, 我主要负责代码的编写. 我设计并实现了可扩展的插件架构, 支持插件的动态加载与停止, 并实现了插件数据的独立管理. 我还使用 PySide6 设计了界面布局, 事件交互逻辑, 与插件系统的无缝集成. 项目中的电费查询插件, 图书馆座位预约插件也是我编写的.

通过此次项目的实践, 尝试编写高可拓展性的代码, 我更加深入地感受到了 python 代码的灵活性; 我也借此机会将校园生活中的信息数据与代码相结合, 感受到了校园网站开发者的辛勤劳动, 得以熟悉 Qt 框架的大致使用方法; 在和其他参与者进行交流时, 我也认识到了应该如何去规范地进行版本控制, 代码管理, 解决团队中不一致的观点.

这些经验都会为我以后的学习生活提供有效帮助.

张梓卫

Thought 五 .2

得益于良好的团队框架，后来使用团队中完善的 `Cache_grabber` 框架，让我能够马上上手研修间预约模块的开发，熟悉了数据处理和 API 调用的结合应用，当然，LateX 课表生成器的开发让我对编译有了更深入的理解，

在不断翻看关卓谦的 `PluginLoader` 的代码到理解，最终写出一份详解的报告从提出到实现，从实现到推翻重来，一切都有迹可循。通过团队交流，我增强了自己的跨领域的开发编程能力，掌握了 Git 多种不同的使用方式，同时跟随团队的项目规范，掌握了如何通过良好的代码架构提高代码复用性和维护性。

同时，我还学到了如何利用 Python 的自动化工具（Selenium-Wire 和 requests）实现复杂的登录流程。翻看华东师范大学开发者文档时，涉及到鉴权，还了解到了 OAuth 2.0 和 JWT 等的认证机制。

是非常愉悦的一次项目经历！

王文锦

Thought 五 .3

在本项目中