# 软件需求分析、架构的 4+1 视图模型和可信性分析研究——以 Yagmail 邮件发送系统为例

卓旭 (212138) 李竞宇 (212115) 李文炜 (212079)

2021 年 11 月

**摘要**

Yagmail [1] 是一个用于进行邮件发送的软件系统，使用 Python 语言编写且开源。其调用方式简单，功能齐全，结果可靠，被许多需要邮件发送功能的 Python 应用所集成。“邮件发送” 对软件可信性的要求很高，如邮件内容格式要正确保留、发送成功率要高、等等。本文以一个略经修改的 Yagmail 的 Fork 为例，进行需求分析，研究软件架构。进一步地，从多方面探究软件架构对软件可信性的影响，并给出相关实验结果，佐证提出的论证。本文的所有相关代码均可在 github.com/z0gSh1u/sa-analysis 取得。

# 目录

# 1 Yagmail 系统需求说明

图 1: Yagmail 软件系统图标

首先简单介绍 Yagmail 的使用方式作为背景。在 Yagmail 的官方文档中，给出了核心功能：邮件发送的调用方式，具体如下。该示例代码首先认证了一个发件邮箱账号，然后组织邮件内容（允许文本、HTML 富文本、附件），最后调用 send 方法，即可向收件人发送邮件。

```
import yagmail
yag = yagmail.SMTP('myUsername', 'myPassword')
contents = ['This is the body, and here is just text http://somedomain/image.png',
          'You can find an audio file attached.', '/local/path/song.mp3']
yag.send('to@someone.com', 'subject', contents)
```

本节接下来，我们将利用 FURPS+ 模型 [2] 以及相关用例，对 Yagmail 系统的系统需求进行分析与说明。

## 1.1 功能性（F）

本小节给出 Yagmail 使用过程中的两个常用用例：登录认证和邮件发送，并通过用例图进一步阐释了用例。

**用例 1：登录认证**

| | |
|---|---|
| **参与者及兴趣列表** | 由发件人启动。在安全，账户信息不泄露的前提下，简单快速的登录认证。 |
| **入口条件** | 安装并导入 Yagmail。 |
| **出口条件** | 发件人成功认证，Yagmail 得到发件地址权限。 |
| **事件流** | 1. 发件人导入 Yagmail 包，并连接网络；<br>2. 发件人调用 register 方法，填入发件地址和密码信息。 |
| **扩展或替代流程** | 2a. 不使用密码，而使用 Keyring 认证；<br>2b. 不使用密码，而使用 OAuth 认证。 |
| **质量需求** | 1. 不受平台、操作系统的影响；<br>2. 认证后安全保存信息，一段时间内不需要重复登录；<br>3. 保证密码数据不被泄露。 |

**用例 2：发送邮件**

| | |
|---|---|
| **参与者及兴趣列表** | 由发件人启动，希望邮件准确无误地发送给收件人。收件人想要接收到内容无误的邮件。 |
| **入口条件** | 发件人已成功认证并连接网络。 |
| **出口条件** | Yagmail 成功发送邮件，收件人收到无误的邮件。 |
| **事件流** | 1. 发件人构造 SMTP 对象，准备发件；<br>2. 发件人调用 send 方法，填入收件地址、主题、正文等信息；<br>3. 收件人成功接收到邮件。 |
| **扩展或替代流程** | 2a. 主题可以为空；<br>2b. 正文可以为空；<br>2c. 邮件可以添加附件。 |
| **质量需求** | 1. 支持群发多个收件人；<br>2. 正文支持 HTML 富文本，图片，纯文本等格式；<br>3. 对非法输入有合理反馈。 |

Yagmail 的用例图如下图所示：

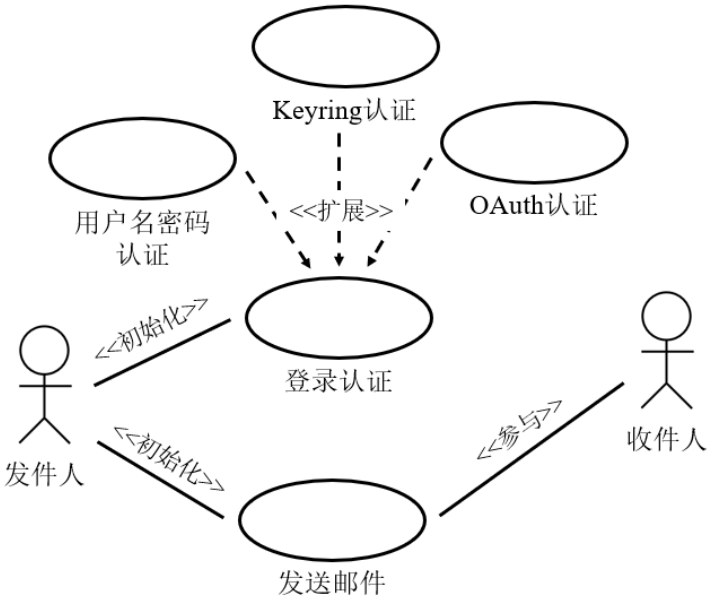


图 2: Yagmail 用例图

## 1.2 可用性 (U)

在安装方面，Yagmail 安装简单，只需使用 Python 自带的包管理工具 pip，执行命令 pip install yagmail[all] 即可安装。对于低版本的 Python（如 2.X 版本），Yagmail 也

提供了垫片代码（Polyfill）与回退方案（Fallback）以支持。

在文档方面，官方提供了详尽的使用文档，介绍了软件的使用细节，以及可能遇到的问题的解决方案与反馈途径。软件整体具有高可用性。

## 1.3 可靠性 (R)

在执行失败率方面，根据 Yagmail 在其他软件系统中的集成使用的反馈，在正确调用的前提下，Yagmail 的执行失败率低。

在可恢复性方面，Yagmail 定义了一系列自定义异常，并对其他可能的异常也做了 try-catch 处理。因局部错误导致整个软件系统崩溃的可能性低，整体可恢复性较高。

在可预测性方面，Yagmail 的代码结构简单，各模块间相互调用链路清晰。通过通读代码，判断 Yagmail 整体执行路径具有较好的可预测性，软件行为也随之具有较好的可预测性。

对于调用链路、执行成功率等方面的具体分析，将在第三节：可信性分析中具体阐述。

## 1.4 性能 (P)

在软件系统的性能方面，在进行邮件发送等基础功能的使用时，Yagmail 系统内部消耗的响应时间很短，这使得 Yagmail 的吞吐量得以提升。另外，Yagmail 库的大小仅约 60KB，且依赖库基本都为 Python 的标准库，所以在其他集成的软件系统中，资源占用率很低。

## 1.5 可支持性 (S)

在可维护性方面，Yagmail 可随时通过 Python 的包管理工具 pip 进行更新，相关错误问题可及时通过 Github Issue 与开发者联系，获得维护修改。

在国际化（i18n）方面，Yagmail 自身支持邮件内容按 UTF-8 编码，在不同语言之间具有较好的国际化兼容性。

## 1.6 其它 (+)

Yagmail 的实现语言为 Python，使用 MIT License 分发。流行、稳健的语言选型增强了 Yagmail 的稳定性，而宽松的 License 选择使 Yagmail 被开源社区更广泛地使用与集成。

# 2 Yagmail 架构的 4+1 视图模型

本节，我们应用 Kruchten 的对软件架构的“4+1”视图模型 [3]，对 Yagmail 系统的架构进行了多角度分析。

## 2.1 逻辑视图

逻辑视图通过使用类图和类模板，体现 Yagmail 系统的不同功能需求。Yagmail 系统通过 yagmail.SMTP 与 smtplib.SMTP 的交互，建立起邮件构建与邮件发送的桥梁。yagmail.SMTP 通过使用地址校验工具类，校验收件发件地址；通过 smtplib.SMTP 进行基于账户密码的验证、基于 Keyring 的认证，或 OAuth 认证，获得发件地址控制权；EmailBuilder 构建邮件传递给 yagmail.SMTP，经处理后交由 smtplib.SMTP 实际发送邮件到收件地址。Yagmail 系统还通过基础设施类保存日志或者错误信息。

Yagmail 的逻辑视图示意图如下图所示：

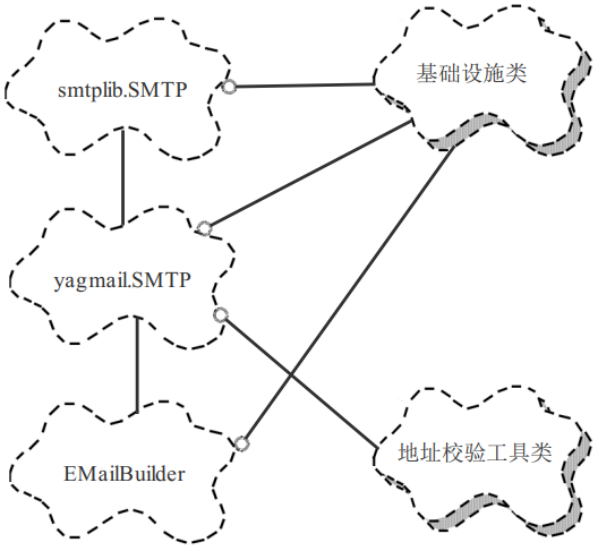


图 3: Yagmail 逻辑视图

## 2.2 过程视图

过程视图给出了 Yagmail 系统使用时的几个主要任务，每个任务是一个独立的控制过程，可以在一个处理节点上独立单独调度，但每次原始调用产生的各过程之间需要串行执行，最终才能完成邮件发送。

Yagmail 的过程视图示意图如下图所示：

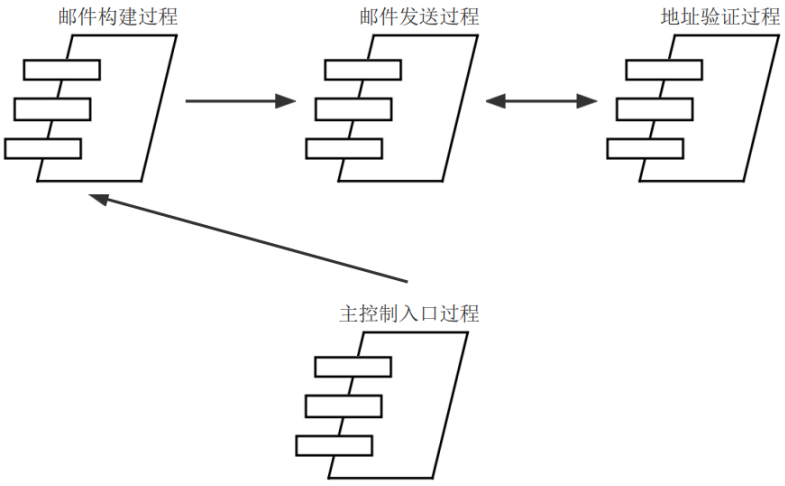


图 4: Yagmail 过程视图

## 2.3 开发视图

开发视图给出了 Yagmail 软件系统各模块的静态组织结构。Yagmail 的子系统层次可分为 5 层：日志、工具、错误模块工作在硬件、操作系统、Python 解释器上，构成了基础设施子系统层次；地址校验模块进一步构成可靠收件地址验证子系统；账密验证、Keyring 验证、OAuth 验证模块进一步构成发件地址权限控制子系统；邮件构建模块进一步构成待发送邮件子系统；邮件发送模块进一步构成 Yagmail 邮件发送系统。

Yagmail 的开发视图示意图如下图所示：


图 5: Yagmail 开发视图

## 2.4 物理视图

物理视图考虑了 Yagmail 系统在运行的过程中，是如何将诸如网络过程、运算过程等各元素映射到具体的物理处理节点上进行运行的。

考虑单机运行情况下的处理核心和作为外部设备的网卡，Yagmail 系统的物理视图示意图如下图所示：

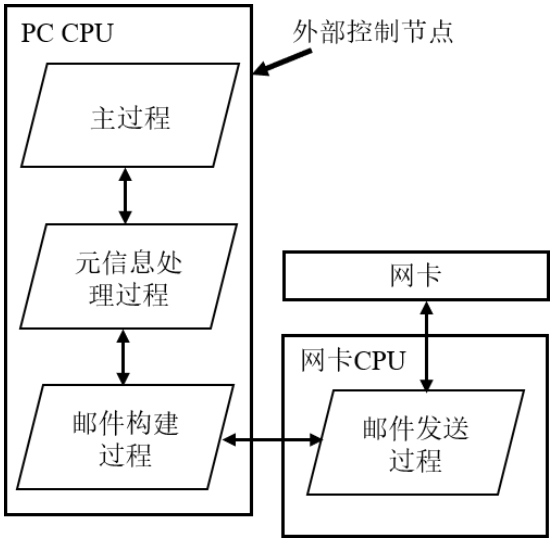


图 6: Yagmail 物理视图

## 2.5 场景

一个场景是一或多个用例的组合。通过场景的组织，可以展示上述四个视图中元素的协调工作模式。我们分析了 Yagmail 最重要的两个需求：发件地址权限的获取、邮件的发送，给出了对应的对象场景图。

对于发件地址权限获取的场景，sender 首先借助地址校验工具类对发件地址进行校验，然后凭地址与登录凭证与 smtp 沟通，从而最终取得对发件地址的发件权限的控制权。

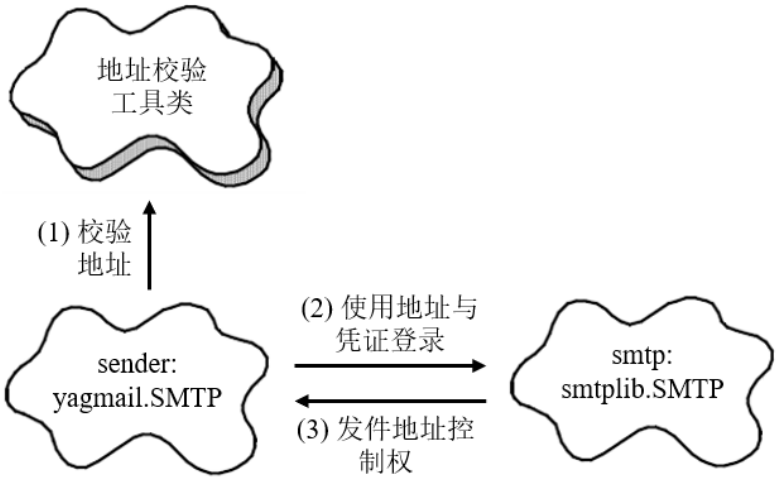


图 7: Yagmail 场景 1：发件地址权限的获取

对于邮件的发送的场景，带有发件地址控制权的参数化的 sender 首先与 builder 合作，完成邮件内容的构建。然后，向 smtp 发起发送邮件的请求。smtp 回传发送邮件的结果，并产生一封邮件（如果成功）。

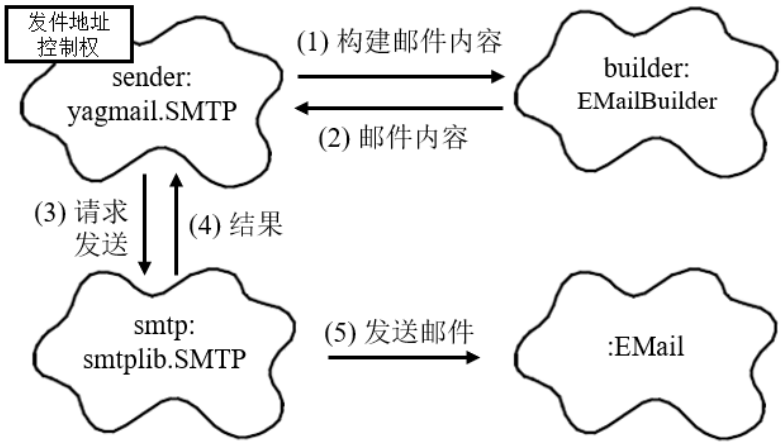


图 8: Yagmail 场景 2：邮件的发送

# 3 Yagmail 的可信性分析

本节，我们对 Yagmail 软件系统的可信性进行了分析评估。首先，我们总结了 Yagmail 在可信性方面应具备的诸项需求。然后，通过对 Yagmail 软件架构、调用路径、程序逻辑与源代码的分析，验证了 Yagmail 具体是如何满足、保障各项可信性需求的。最后，借助一些在软件可信性分析方面进行更具体的量化分析的方法，从定性走向定量，对 Yagmail 的可信性进行了进一步的度量。

软件可信性是一个新鲜的概念。在对可信性进行具体分析之前，有必要适度明确可信性的概念。结合 [4] 和 [6] 对软件可信性的定义，我们认为“可信性”指的是软件系统在实现既定功能时，行为和结果是可以预测的、可以控制的。即使是在受到各种外部干扰（特殊情况）时，也具备较高的容错性。为了提高软件的可信性，需要在可用性、功能、可靠性（容错）、安全性（机密性、完整性）、性能（响应时间、资源消耗）、维护代价等多方面共同优化。

## 3.1 可信性需求

本小节，我们将对 Yagmail 在可信性方面应具备的基本需求进行列举说明，具体如下。

1.（高可用性）当正确调用 Yagmail 发送邮件时，邮件发送的成功率要高。

2.（高可用性）进行登录认证时，在信息输入正确的情况下，登陆成功率要高。

3.（高容错性）当登陆或邮件发送失败时，需要给用户反馈，提示错误原因，并有相应的错误处理机制和记录。

4.（高容错性）对用户填写的邮件地址，需要验证合法性。

5.（高安全性）进行登录认证时，必须保证用户隐私信息不被泄露，

6.（高安全性）邮件发送到指定的收件人，而不是第三方。

7.（高安全性）邮件内容格式要保留完整。

8.（性能）邮件发送的响应时间要快，资源占用要低。

以上为 Yagmail 应满足的主要的可信性需求。

## 3.2 可信性保障

本小节，我们将对 Yagmail 具体是如何满足、保障上一小节所提到的各项可信性需求，进行分析说明。在具体分析 Yagmail 的可信性实现之前，有必要对它的各组件的依赖关系进行说明。我们借助 Pyan 工具 [7]：一款针对 Python 语言的静态分析、组件/函数调用图以及“define-use”链路绘制工具，对 Yagmail 进行组件图分析绘制。绘制层级到组件级而非函数级，图中的箭头表示“use”关系，“define”关系不在图中体现。

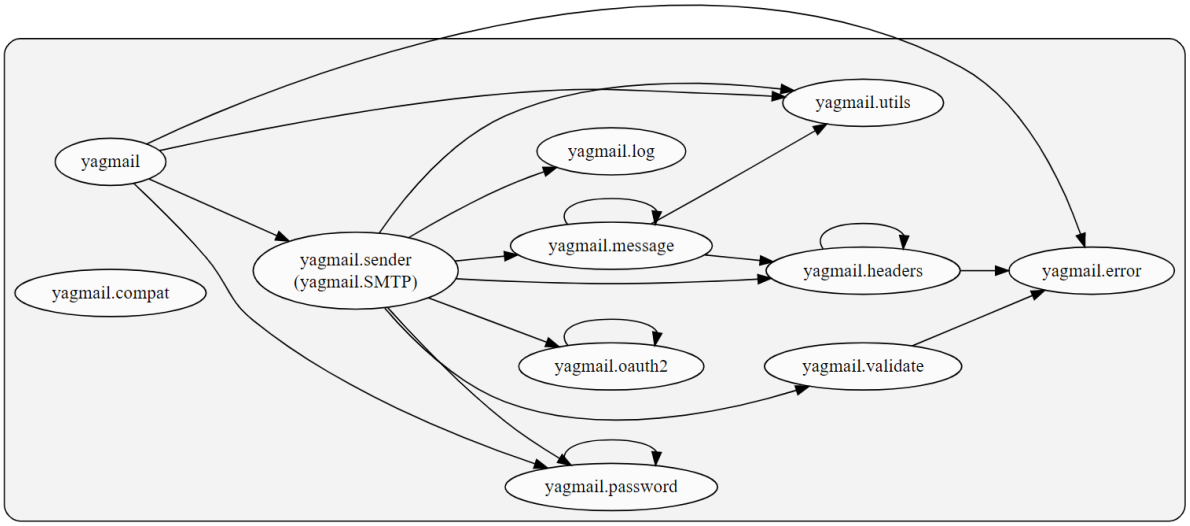


图 9: Yagmail 的组件依赖图

### 3.2.1 高可用性

为了对高可用性进行保障，在发送邮件的成功率方面，Yagmail 第一次发送失败时，会再尝试两次间隔 3、6 秒的重新发送，以提高发送的成功率。如果仍然失败，则记录下来。从软件架构层面看，结合前文的开发视图，本部分职责由顶层组件 yagmail.sender 完成，相关代码如下：

```
while attempts < 3: # 最大尝试次数为3
    try:
        result = self.smtp.sendmail(self.user, recipients, msg_string)
        self.log.info("Message sent to %s", recipients)
        return result # 发送成功，直接返回
    except smtplib.SMTPServerDisconnected as e: # 发送出现异常
        self.log.error(e)
        attempts += 1
        time.sleep(attempts * 3) # 间隔一段时间重新发送
self.unsent.append((recipients, msg_string)) # 仍然失败，保存相关信息
```

### 3.2.2 高容错性

对用户填写的邮件地址，结合前文的开发视图，Yagmail 的较低层次的 yagmail.validate 组件会使用基于 RFC 2822 标准构建的正则表达式，验证邮件地址的格式。如果不符合要求，则会报错提示不符合 RFC 2822 的标准。但无法验证该邮件地址是否为实际注册存在的邮件地址。

```
def validate_email_with_regex(email_address):
    if not re.match(VALID_ADDRESS_REGEXP, email_address): # 使用正则表达式验证
        emsg = 'Emailaddress "{}" is not valid according to RFC 2822 standards'.format(email_address)
        raise YagInvalidEmailAddress(emsg) # 发起异常
```

### 3.2.3 高安全性

在登陆时，除了明文账号密码方式以外，Yagmail 还提供更安全先进的 OAuth 验证方式，对 GMail 发件邮箱可使用 Google 授权口令授权，起到了保障用户邮箱、密码等隐私信息的作用另外，Yagmail 还提供基于操作系统 Keyring 的认证方式，存储密码信息。具体做法是使用 register 方法，其内部调用 keyring 库，将相关隐私信息存入操作系统。以上这些在软件架构层面，结合前文绘制的开发视图和逻辑视图，主要由中间层次的发件地址权限控制类模块实现。它们利用了底层模块提供的功能，并对更高层次的模块给与权限与安全保障。

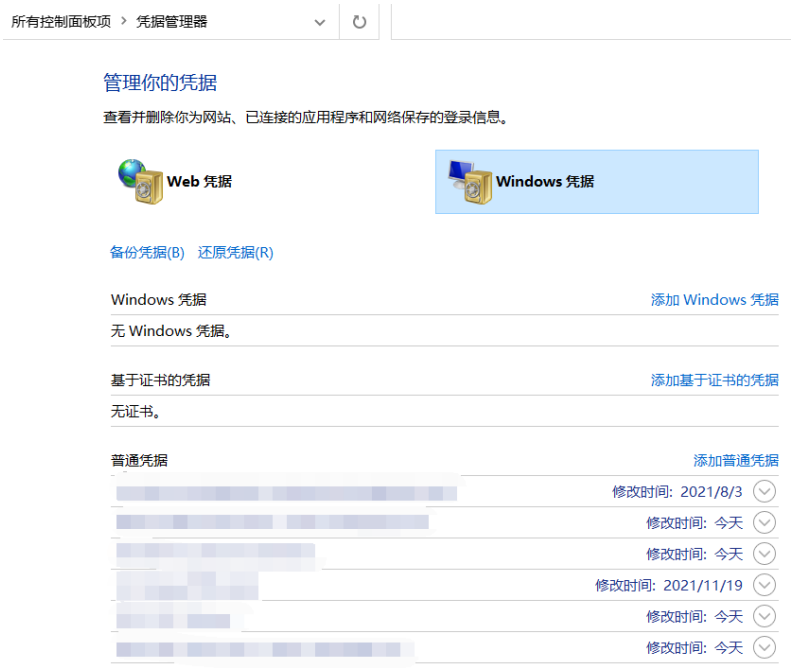


图 10: Windows 系统的 Keyring 服务：凭据管理器

上述步骤的相关代码如下所示：

```
yag = yagmail.SMTP("user@gmail.com", oauth2_file="~/oauth2_creds.json") # OAuth认证方式
yagmail.register('myUsername', 'myPassword') # 登记Keyring信息
keyring.set_password('yagmail', 'myUsername', 'myPassword') # 内部调用keyring库实现
```

### 3.2.4 性能

为了提高 Yagmail 的整体性能，在执行时间方面，Yagmail 的组件依赖关系简单，调用路径短，这使得执行的效率得到提高；在资源占用方面，Yagmail 的依赖库绝大部分都是 Python 内部的标准库，整体库大小控制在了 60KB 左右，资源占用低，更易于集成到其他软件系统。对于 Yagmail 性能的测试，将在下一小节更具体地介绍。

## 3.3 可信性度量

本小节，我们按照 [5] 中提供的基于可靠概率法的可靠性度量方法，以及使用圈复杂度和扇入扇出度的可维护性度量方法，对 Yagmail 的可信性在这两个方面进行度量，以获得更细致的定量的分析结果；使用插桩（Stub）Mock 法，对 Yagmail 进行了性能度量；按照 [8] 提出的基于用户评价、迭代和缺陷跟踪的开源软件可信性评价方法，阐述了一种对 Yagmail 在用户侧的可信性进行度量的做法。在整体流程中，我们还借鉴了 [9][10] 中的一些思路。

### 3.3.1 可靠性度量：使用可靠概率法

可靠概率 RP 的度量结果为一个实数值，用来表示软件系统正常运行时软件架构不会发生错误的概率。为简化分析，我们以分析 Yagmail 的场景视图的场景 2：邮件的发送为例。在 UML 模型中，场景由 UML 顺序图表示。因此，首先将图 8 重绘为 UML 顺序图，如下图所示：

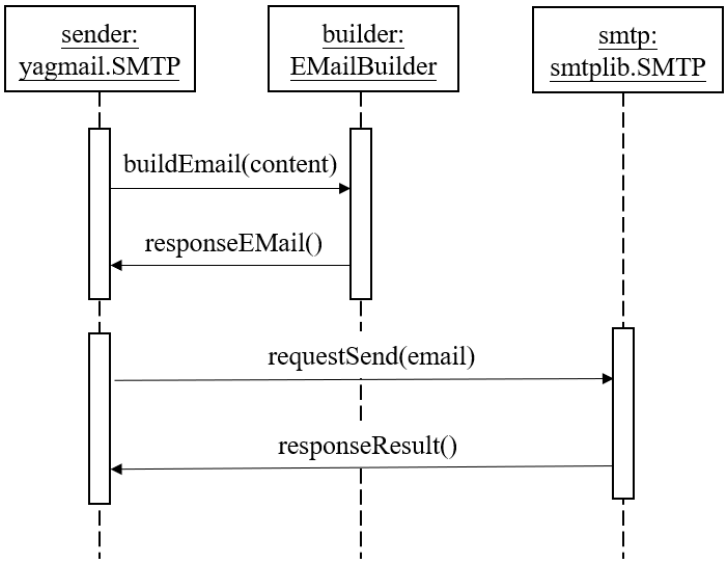


图 11: “场景 2：邮件的发送”的 UML 顺序图

首先考虑组件对场景可靠性的影响。将所使用的的三个组件 $i$ 和一次执行平均可能故障概率 $\text{component}_i$ 列表如下：

表 1: “场景 2：邮件的发送”的组件平均可能故障概率

| **组件** $i$ | $\text{component}_i$ |
|---|---|
| yagmail.SMTP | 0.5 % |
| EMailBuilder | 0.2 % |
| smtplib.SMTP | 0.1 % |

对于组件 smtplib.SMTP，由于其为 Python 语言标准库的一部分，故认为其故障概率很低，为 0.1%。对于 Yagmail 内部的组件，由于没有运行时的遥测（Telemetry）回传数据，我们依据先验知识对故障概率进行设定。EMailBuilder 的职能相对简单，故赋较小的故障概率；yagmail.SMTP 的工作内容较为复杂，外部调用多，故赋较大的故障概率。

其次考虑连接件对场景可靠性的影响。由于对物理连接件，如网卡、电缆、光纤等硬件和介质的失败概率分析已超出对 Yagmail 的软件架构进行分析的范畴，故我们取物理连接件的失败概率 $\text{connector}_j = 0$。

综上，结合各组件在场景中的被调用次数 $x_i$，我们可计算出该场景的可靠概率：

$$\begin{aligned} \text{ssp} &= p_{\text{component}} \times p_{\text{connector}} \\ &= \Pi_i (1 - \text{component}_i)^{x_i} \times 1 \\ &= (1 - 0.005)^2 \times (1 - 0.002)^1 \times (1 - 0.001)^1 \times 100\% \\ &= 98.7\% \end{aligned} \tag{1}$$

可见，在当前的软件架构下，Yagmail 有效控制了各组件的被调用次数，以达到对整体失败概率进行控制的效果，系统整体可靠概率高。

### 3.3.2 可维护性度量：使用圈复杂度和扇入扇出度

在软件架构度量和评估中，圈复杂度表示组件图中由模块依赖关系组成的依赖控制路径的数量；而扇入指的是直接调用该模块的上级模块的数量；扇出指的是该模块调用下级模块的数量。

我们选择对 Yagmail 的核心组件：Yagmail.SMTP 进行可维护性度量。结合图 9 绘制的各组件之间的依赖关系图，根据与 Yagmail.SMTP 相关的组件提取出来构成的子图，以及对源代码中 import 关系的分析，可得到如下表所示的数据：

表 2: Yagmail 可维护性度量数据

| 项目 | 符号 | 值 | 说明 |
|---|---|---|---|
| 组件图中所有外部组件的数目 | $totalN$ | 9 | 顶层包、log、message、oauth2、password、utils、headers、validate、error |
| 组件图中所有外部组件的关联关系 | $totalE$ | 15 | 不含组件内部的关联关系，即去掉内部关联边的总边数 |
| 层数 | $L$ | 2 | UML 组件图数目，仅有 compat 组件游离在与 yagmail.SMTP 相关的子图外 |
| 该组件对其他组件的依赖 | $E$ | 7 | 出边数量 |
| 该组件提供的接口数 | $W$ | 2 | - |
| 其他组件对该组件的依赖 | $X$ | 2 | 仅顶层包中存在 |
| 该组件调用其他组件的接口数 | $R$ | 9 | - |

由上表列出的数据，代入相关公式，可计算出圈复杂度:

$$\begin{aligned} \mathrm{CCN} &= (totalE - totalN) + 2L \\ &= (15 - 9) + 2 \times 2 \\ &= 10 \end{aligned} \tag{2}$$

圈复杂度小于等于 10，是一个较为适宜的值。

同理可计算出扇入扇出度:

$$\begin{aligned} \mathrm{FFC} &= \mathrm{CCN} \times ((E + W) \times (X + R))^2 \\ &= 10 \times ((7 + 2) \times (2 + 9))^2 \\ &= 98010 \end{aligned} \tag{3}$$

其中扇入指标主要由 $W$、$X$ 表现，扇出指标主要由 $E$、$R$ 表现，可以看出，由于 Yagmail.SMTP 是一个偏顶层的模块（正如“图 5: 开发视图”所示），所以表现出扇出比较大的特性，符合优秀、典型的软件架构的特点。

Yagmail 在可维护性度量方面的优秀、典型的指标，恰说明了 Yagmail 的软件架构的优秀性。因为只有适当组织软件架构各层级、各模块之间的依赖、调用关系，才能获得好的圈复杂度和扇入扇出相关指标。

### 3.3.3 性能度量：使用插桩（Stub）Mock 模拟法

在软件测试领域，代码插桩（Stub）技术是一种常用的测试辅助技术。它使得测试人员在对软件的一部分进行测试时，可以为依赖但还未完成的部分暂时用一段代码“占坑”，以临时切断这种依赖关系，使得测试得以继续。这种技术的一种实现是 Mock 模拟方法，即通过随机数据，模拟多种场景。该部分的实验代码在 yagmail-stub-mock 目录下。

由于 Yagmail 的登陆和邮件发送功能的流程与邮件发送服务器强相关，而倘若构造一系列真实的邮件发送情景，实验成本较高，且大部分邮件服务提供商对此类通过脚本大规模发送邮件的行为，都有相应的风险控制策略，容易触发警报。因此，我们使用模拟数据接口对 Yagmail 的邮件发送功能（yagmail.SMTP.send）进行性能测试，以简化测试流程，规避上述问题。

我们使用 Mock 技术，融入先验知识，模拟两部分内容：

一是 Yagmail 与远端邮件发送服务器进行通讯时的时间消耗（往返时延）$RTT$，其模拟值为从正态分布 $N(\mu, \sigma) = N(1.5, 0.5)$ 的采样，且要求 $0 < RTT < 3$，单位为秒。这表示在邮件发送时网络通讯部分的耗时绝大部分落在 1 到 2 秒的范围内，最大不超过 3 秒，这符合现有主流个人邮件服务提供商侧的性能水平。对得到的 $RTT$，在进行邮件发送时按该时间阻塞程序，以模拟 Yagmail 对服务器响应的等待。

二是 Yagmail 在进行邮件发送时邮件发送服务器的发送成功概率。其生成方式为，当从均匀分布 $U(0, 1)$ 上采样的一随机值 $p > 0.5\%$ 时，为发送成功，即发送失败率约为 $0.5\%$，这符合现有主流个人邮件服务提供商侧的成功率水平。当模拟发送失败时，即会触发 Yagmail 的高可用性重试逻辑。

以模拟内容一为例，在完成代码 Mock 前后，代码的变异如下：

```
# Mock前：需要与外部服务器联系
result = self.smtp.sendmail(self.user, recipients, msg_string)

# Mock后：切断了外部依赖
waitTimeSec = randomInRangeGauss(0, 3, 1.5, 0.5)
self.slpTimesTot += waitTimeSec # 记录总RTT
time.sleep(waitTimeSec) # 模拟等待
```

固定仅需执行一次的登录操作的时间消耗为 3 秒，模拟发送 500 封邮件。测试过程中，Yagmail 承受住了串行压力，面对发送失败的错误也没有发生崩溃。测试过程图、相关代码如下所示：


图 12: 邮件发送性能测试项目测试过程图

```
# 求初始化时间开销
tic = time.time(); smtp = yagmail.SMTP(user='MOCK'); toc = time.time() + LOGIN_TIME

# 求模拟500封发送的时间开销
tic = time.time()
for i in tqdm(range(SEND_COUNT)):
    smtp.send()
toc = time.time()
```

通过整理数据，可得到性能测试结果表如下：

表 3: 性能测试结果表

| 项目 | 耗时（秒） | 项目 | 值 |
|---|---|---|---|
| 初始化与登陆耗时 | 3.008 | 邮件发送失败数 | 3 |
| Yagmail 初始化耗时 | 3.008-3=0.008 | 邮件发送成功率 | 99.4 % |
| 邮件发送总耗时 | 748.237 | 程序崩溃率 | 0 |
| 总 RTT | 733.736 | （下空） | |
| 邮件发送 Yagmail 耗时 | 748.237-733.736 =14.501 | | |
| 平均每封 Yagmail 耗时 | 14.501/500=0.03 | | |

可见，略去与网络、服务商等外界因素相关的耗时，Yagmail 在邮件发送上消耗的时间极少，整体性能很高，且在快速的同时，保持了较高的成功率。这样优秀的表现与 Yagmail 优秀的软件架构是分不开的。这验证了我们在 3.2.4 节的判断，并且我们将在第四节的实验中进一步分析、证明这一点。

### 3.3.4 开源软件用户侧可信性度量：基于用户评价、迭代和缺陷跟踪

通过文献阅读，我们从 [8] 了解到，对于开源软件，可以通过爬取缺陷跟踪系统中的可信证据，对可信属性之间的联系进行分析，评价组件和开源软件自身在运行时的可信性。而 Yagmail 在 GitHub 平台开源的特点，使得它天生就具备 Issues 这一缺陷追踪

渠道，且可从中获知用户评价数据；它还天生具有 Pull Request 这一迭代追踪渠道，可以了解各缺陷的迭代修复过程。这些数据让我们能够利用该文献中的方法，对 Yagmail 的用户侧可信性进行度量。

由于该文献中提到的方法实现过程较为复杂，且所需数据需要爬取、仔细清洗后才可开始分析流程，因此本文此处仅做介绍提及。

# 4 实验：Yagmail 软件架构的变异与对比

为了进一步说明 Yagmail 的软件架构设计对软件可信性的影响，本节将对 Yagmail 在软件架构进行一定的变异、修改，引入架构坏味道。接着，将前后版本的 Yagmail 进行对比，发现架构变异对软件可信性的负面作用。本部分实验的相关代码在 yagmail-bad-smell 目录下。

## 4.1 实验实施

根据 [5] 的叙述，软件架构层面的坏味道是一种设计级的坏味道，其抽象层次更高。本文对 Yagmail 变异出如下三个种类的架构坏味道：组件嫉妒（Component Envy）、过度分散的功能（Scattered Functionality）、模糊接口（Ambiguous Interface）。

**为形成组件嫉妒**，我们将验证登录环节的 OAuth 型验证方式的实现交由 yagmail.SMTP 内部实现，取消对 yagmail.oauth2 的导入。这样，增加了 yagmail.SMTP 组件的功能。

**为变异出过度分散的功能**，我们将验证邮件地址的合法性的 yagmail.validate 模块的核心功能 validate_email_with_regex 复制一份到工具模块 yagmail.utils 中，因为该功能也符合工具功能或辅助功能的定义。但在 yagmail.validate 模块中的实现，该功能认为即使没有点号，邮件地址也可能是合法的，因为当用户是在某个局域网、使用自建 DNS，或者修改了 Hosts 文件的情况下，上述邮件地址确实可能被解析。

**为了引入模糊接口**，我们允许用户在调用 yagmail.SMTP 模块的核心功能 send 发送邮件时，在“主题”（Subject）一项参数中填入数字值。如此，在确定邮件主题时，需要加一步对数据类型的判断，如果为数字值，需要先将其转换为字符串表示。该功能是有意义的，因为一些通知类或内部信息邮件的主题，可以用纯数字表示 ID 编号，以便归档。

在完成上述变异流程后，我们使用与第三节类似的软件可信性的分析方式，对变异版本的 Yagmail 重复可信性分析，并进行对比。相关结果与分析于下一小节展示。

## 4.2 结果与分析

在完成上一小节提到的变异修改后，yagmail.SMTP 模块的内部调用链路图如下图所示（使用 Pyan 绘制）：

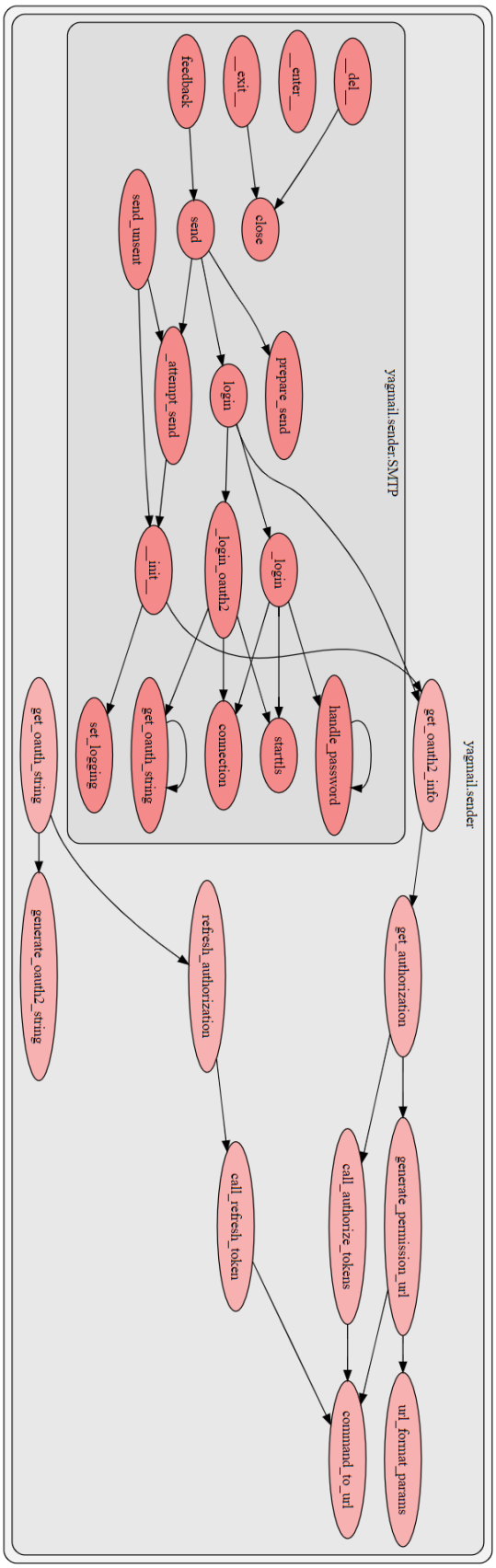


图 13: 变异后的 yagmail.SMTP 模块的内部调用链路（经旋转以清晰展示）

**组件嫉妒**变异导致了函数调用路径的变化，OAuth 认证不再需要涉及其他模块，这似乎降低了 yagmail.SMTP 组件的圈复杂度，但该架构违背了单一职责原则，内部调用链路复杂了许多，进一步地导致了组件的可测试性、可理解性和可复用性的降低：可理解性的降低体现在 yagmail.SMTP 模块既要负责邮件的发送，又要负责发件地址权限的控制，这导致“图 5：开发视图”中的第 3 层和第 5 层混杂在一起，如下图所示。这破坏了分层结构，难以总结、理解该模块实现的功能；可复用性的降低体现在模块的体积更大，引入的成本更高，并且低可理解性也加大了引入的心智负担；可测试性的降低是因为模块级的测试失败时，需要更多的工作用来确定具体是哪些函数或者哪条调用链路产生的问题。

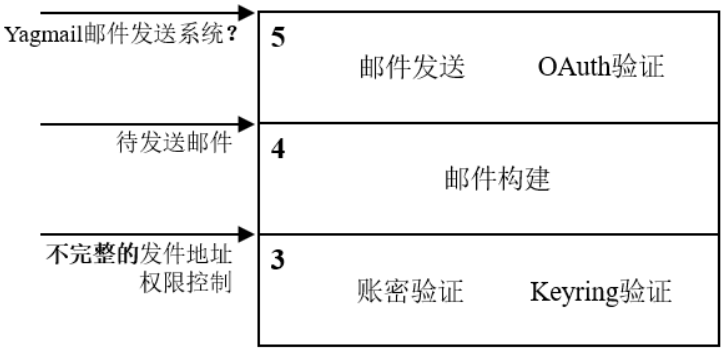


图 14: 组件嫉妒下的开发视图变化

**过度分散的功能**使得组件的关注点不能很好地分离。关注点对组件的多重性，即组件对关注点的非正交性，会导致软件架构被破坏。对于需要使用同一功能的不同高层组件，可能会挑选不同的低层组件提供服务。则逻辑视图下连接关系将变多、变复杂，新加入的模块难以根据原有视图信息选择合适的低层模块。如下图所示，根据前文的“图 3：逻辑视图”可知，yagmail.SMTP 对基础设施类（如 yagmail.util）和地址校验工具类（yagmail.validate）都有逻辑关联，此时其对 validate_email_with_regex 功能的选择就出现二义。而在测试与维护方面，关注点向组件的不恰当对应还使得代价变大，需要对更多的组件进行测试与维护。

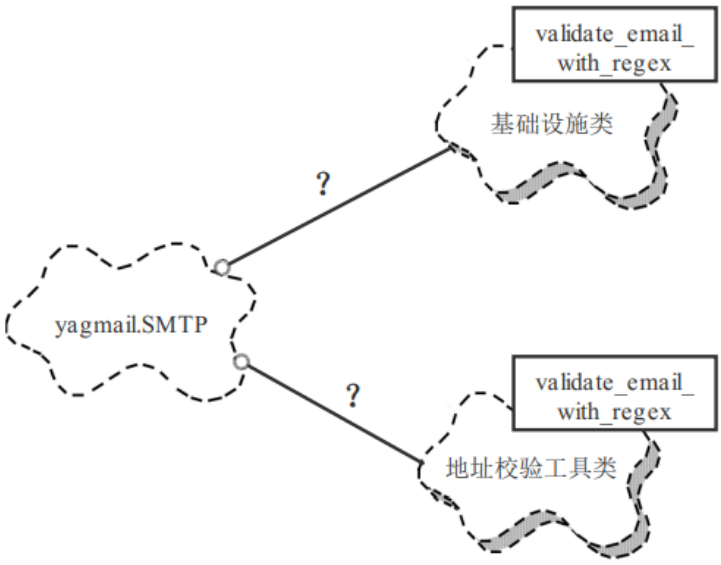


图 15: 过度分散的功能导致关注点难以分离

**模糊接口**带来了应用性能的降低。程序需要对传入的数据的类型在运行时进行判断，必要时进行相应的转换。对于 Python 此类强类型动态类型语言，需要增加运行时类型判断与显式转换。倘若对于如 C++ 这类静态类型语言，则还需要通过反射等运行时类型收集技术，妥善处理类型；对于 JavaScript 这类弱类型动态类型语言，类型转换过程经常成为安全漏洞的发生处。因此，无论是从运行的速度，还是运行的安全性来看，模糊接口对程序性能都有负面影响。

综上所述，通过实验和后续的具体分析，我们可以得出引入架构坏味道对软件可信性存在负面影响，进一步说明了好的软件架构对软件可信性的重要性。

# 5 总结

在本课程作业中，我们首先通过 FURPS+ 模型对 Yagmail 软件系统进行了多方面的需求分析说明，以对该软件系统的整体功能形成认识与把握。接着，我们应用 Kruchten 的“4+1”视图模型，对 Yagmail 软件系统的软件架构进行了多角度的观察与分析。

在第三节，我们进一步应用从源码和架构层面的定性分析，以及基于度量的分析方式，对我们提出的 Yagmail 应具备的数项可信性需求的具体保障方式以及保障效果进行了分析说明。在第四节，我们则通过引入软件架构坏味道，对 Yagmail 的软件架构进行变异，以及可信性方面的对比分析。实验和分析表明，Yagmail 对软件的可信性进行了行之有效的保障工作，整体可信性高，特别是软件架构的设计，对可信性起到了重要作用。在对软件架构进行负面的变异后，软件的可信性确实发生了降低，这更说明了好的软件架构对软件可信性的重要性。

通过本次实验，我们对课堂所学知识有了更深入的消化吸收，并进行了实操应用。特别是通过对 Yagmail 软件功能、结构和使用的分析，对需求的 FURPS+ 分析模型，以及软件架构的“4+1”视图模型的含义有了深入的了解，让之后的应用也更加得心应手。而在后面关于可信性的分析实验中，我们深刻体会到了软件可信性的重要性，以及好的软件架构对可信性的关键性，这将极大地指导我们日后的软件开发工作。

# 参考资料

[1] kootenpv. Yagmail [CP/OL]. 2021. https://github.com/kootenpv/yagmail.

[2] Grady Robert, Caswell Deborah. Software Metrics: Establishing a Company-wide Program [M]. Prentice Hall. 1987.

[3] Philippe Kruchten. Architectural Blueprints—The “4+1” View Model of Software Architecture [J]. IEEE Software. 1995.

[4] 刘克. “可信软件基础研究”重大研究计划综述 [J]. 中国科学基金. 2008..

[5] 李必信, 廖力, 王璐璐 等. 《软件架构理论与实践》[M]. 机械工业出版社. 2019.

[6] 李必信. 《软件开发方法与技术》课程课件 [EB/OL]. 2021. （内部资料）.

[7] Technologicat. Pyan [CP/OL]. 2021. https://github.com/Technologicat/pyan.

[8] 孙晶, 刘丽丽. 开源软件可信性评价方法 [J]. 计算机工程与设计. 2017.

[9] 吴晓娜, 李必信, 刘翠翠 等. 基于多维质量属性偏好的可信服务选择 [J]. 计算机应用与软件. 2012.

[10] 王德鑫, 王青, 贺劼. 基于证据的软件过程可信度模型及评估方法 [J]. 软件学报. 2017.