Modulon Labs 技术笔试题

核心考察能力

本次笔试重在考察你的综合能力，我们尤其关注以下三个层面：

1. **Vibe Coding 基础能力**: 你是否能与 AI 高效协作，利用 AI 工具（如 `Gemini CLI`）来探索问题、生成代码并解决实际问题。
2. **抽象与自动化思维**: 你能否清晰地定义问题边界，将任务拆解为明确的规则和步骤，从而让 `Gemini CLI` 和自动化脚本能接管大部分执行工作。
3. **工程化交付能力**: 我们不仅看重最终的交付成果，同样关注整个过程的“优美性”——包括清晰的文档、规范的版本控制、可复用的代码结构，以及你的方案在应对未来类似任务时的通用潜力。

考试引言

你好！欢迎参与本次技术笔试。

我们深知，这份笔试题中可能会包含一些你初次接触的新概念或工具，但这正是挑战的乐趣所在。请不要畏惧，我们鼓励你大胆尝试，勇敢地去理解问题背后的本质，并进行抽象思考。我们相信，无论最终结果如何，完成这次笔试的过程本身，都将为你带来一次宝贵的、甚至是突破性的成长体验。

本次笔试旨在考察你的实际编码能力、AI 工具应用能力以及自主学习和解决问题的能力。我们期望你不仅能完成编码任务，更能展示出在AI协同工作流下的探索精神和工程化思考。

核心任务

实现一个能够批量生产宣传素材的自动化脚本。

具体场景：假设你需要为一系列线上活动制作宣传图，每张图的背景、版式都相同，只有文字内容不同。你需要开发一个脚本，根据提供的文本，自动生成符合要求的图片素材。

> **重要提示**：本次任务需要生成的素材样式，请参考此链接： 37 大康讲SOP发布了一篇小红书笔记，快来看吧！ 😆 2aWhFDU7Pev 😆 http://xhslink.com/m/8xp6MN1zvPe 复制本条信息，打开【小红书】App查看精彩内容！ 。在开始前，请仔细观察并总结参考图片中的核心排版要点（如字体、字号、间距、颜色等），这会是 TDD 流程中重要的对比基准。另外你也可以基于此，去做进一步的格式改进，也是可以的。

详细要求

1. 功能要求

- **输入**: 脚本接受一个或多个文字文本作为参数。
- **处理**: 脚本能在一个预设的网页模板上，将输入文本填充到指定位置。
- **输出**: 脚本或者MCP能将生成的网页内容截图，并保存为特定尺寸的图片文件。最终产出一个包含所有生成图片的文件包。

2. 技术要求

- **核心工具**: 必须使用 `Gemini CLI` 作为主要的编码和执行环境。
- **自动化浏览器**: 必须使用 `Playwright MCP` 或其他你熟悉的浏览器 `MCP (Managed Component Protocol)` 来操作网页、填充文本和截图。
- **开发流程**:
  - **网页枢纽**: 整个工作流需要以一个本地或公开的网页文件为核心。具体实现路径为：通过脚本将指定文本动态填充到该网页文件中，然后利用 `Playwright MCP` 对更新后的网页进行截图，以此方式生成最终素材。
  - **自动迭代**: 请在开发过程中让 Gemini CLI 自动对比生成的图片与要求的图片，识别差异并进行迭代优化，以确保脚本的稳定性和正确性。
- **版本控制**: 项目须使用 `Git` 进行版本管理，并将最终代码上传到你自己的 GitHub 仓库。

3. 交付物要求

1. **代码仓库**: 一个公开的 GitHub 仓库链接，其中应包含以下内容：
   - **自动化脚本**: 能够完成核心任务的脚本文件。我们鼓励实现端到端的完全自动化。
   - **操作手册 (`manual.md`)**: 详细说明如何配置环境和运行脚本。
   - **（可选）Gemini CLI 执行文档 (`cc-runner.md`)**: 如果你的方案包含部分手动步骤或需要 `Gemini CLI` 介入执行（例如调用 `MCP` 截图），请提供此文档。它应包含清晰的指令，能让 `Gemini CLI` 读取并完成整个任务流程。
   - **研发日志 (`dev_log.md`)**: 在开发过程中，要求 `Gemini CLI` 记录研发日志。请在此文档中提供相关指令，以便 `Gemini CLI` 能够自动记录开发过程中的关键步骤和思考。
   - **心得文档 (`experience.md`)**: 记录你的开发过程、思考和总结。

验收标准

- **功能完整**: 脚本可以无错运行，并根据输入文本正确生成符合尺寸要求的图片文件包。
- **技术合规**: 明确使用了 `Gemini CLI` 和 `Playwright MCP` ，并能在提交的心得文档中体现出来。
- **代码质量**: 代码结构清晰，注释必要，易于理解和维护。
- **版本历史**: Git 提交历史清晰、规范，能够反映开发过程。
- **文档质量**: 心得文档内容充实，思考深入，能展现你的学习和总结能力。

资源与指南

1. Gemini CLI 安装与配置

`Gemini CLI` 的安装和配置是完成本次笔试的基础。详细教程请参考以下链接：

请移步查阅：Gemini CLI 教程

请务必按照该指南完成所有配置，再继续后续步骤。

2. Playwright MCP 配置

Playwright MCP 是一个强大的浏览器自动化工具，它允许 `Gemini CLI` 通过指令来控制浏览器行为。详细的配置和使用方法，请参考以下官方文档：

- **Gemini CLI - MCP Server 文档**: https://github.com/google-gemini/gemini-cli/blob/main/docs/tools/mcp-server.md
- **Playwright MCP 原始仓库 (重点)**: https://github.com/microsoft/playwright-mcp

> **重要提示**：我们强烈建议你**重点阅读第二个链接（Playwright MCP 原始仓库）**中的 `README.md` 文件。它详细说明了所有可用的浏览器操作指令（如截图、输入、点击

> 等）及其参数，这对于你完成本次任务至关重要。

3. “自动迭代”与测试驱动开发（TDD）哲学

我们提到的“自动迭代”，本质上是测试驱动开发（TDD）思想在 AI 协同工作流中的一种应用。TDD 的核心循环是“红-绿-重构”，可以这样理解：

- **红（Red）**：先失败。针对你的目标，首先创建一个自动化测试，用于验证最终产出的图片是否符合预期。例如，你可以先准备一张“目标图片”，然后让测试脚本去对比 `Gemini CLI` 生成的图片和这张目标图片。由于代码还没写，这个测试一开始必然是失败的。
- **绿（Green）**：求成功。接下来，驱动 `Gemini CLI` 编写最少的代码，让上面的测试通过。目标是尽快让生成的图片与“目标图片”在像素层面完全一致，使测试变“绿”。
- **重构（Refactor）**：做优化。在测试通过的前提下，再回过头来优化和重构你的代码、脚本或指令（Prompt），使其更健壮、更优雅、更高效，同时确保测试依然是“绿”的。

> **给你的提示：**
>
> 在这个任务中，你可以指示 `Gemini CLI` 遵循这个流程：
>
> 1. **定义目标**：首先给 `Gemini CLI` 一张完美的“参考截图”。
> 2. **创建测试**：可以让 `Gemini CLI` 编写一个简单的脚本或使用工具来对比两张图片（它生成的 vs 你给的参考）的差异，甚至也可以直接让 `Gemini CLI` 自行对比，因为大模型具备对比图片的能力。
> 3. **迭代开发**：命令它不断修改生成图片的脚本，直到图片对比测试通过为止。
> 4. **最终交付**：交付的脚本应该是这个迭代过程最终的、成功的版本。
>
> 这种方式不仅能保证产出质量，更能体现出一种严谨、闭环的工程化开发思维。

4. 利用文档实现“记忆”与“日志”

`Gemini CLI` 本身是无状态的，它不会“记住”上一次复杂任务的上下文。为了解决这个问题，我们可以巧妙地利用本地文件作为它的“外部记忆体”。

核心思路：将需要长期遵循的指令、中间状态或历史记录写入文档，然后在需要时，让 `Gemini CLI` 读取这些文档来“回忆”起上下文或接下来的任务。

具体技巧：

1. **制定行动指南 (`cc-runner.md`)**：

   你可以创建一个 `Markdown` 文件，在里面用自然语言或伪代码写下清晰、分步骤的行动计划。然后，你可以让 `Gemini CLI` 读取这个文件，并像执行任务清单一样去完成工作。

   - **示例指令**：“`Gemini CLI` ，请读取并执行 `cc-runner.md` 中的所有步骤。”
   - **好处**：这使得复杂的任务流程变得清晰、可重复和易于调试。

2. **记录研发日志 (`dev_log.md`)**：

   在交付物中，我们要求你提交一份研发日志。你可以通过给 `Gemini CLI` 下达一个初始指令来实现这一点，让它在后续的每一步操作后，都将自己的思考、执行的命令、遇到的问题和解决方案追加（append）到 `dev_log.md` 文件中。

   - **示例指令**：“`Gemini CLI` ，在接下来的任务中，请将你的每一个步骤和思考过程都记录到 `dev_log.md` 文件里。”
   - **好处**：这不仅能满足交付要求，更能帮助你追踪和复盘整个开发过程。

3. **默认规则 (`gemini.md`)**： `gemini.md` 是一个关键文档，默认情况下每次任务开始时都会被调用。它包含了 `Gemini CLI` 的核心行为准则和全局指令，确保其操作符合预期。
   - **自动加载**：在每次任务启动时， `Gemini CLI` 会自动读取 `gemini.md` ，以确保其行为与项目的基本要求一致。
   - **全局指令**：该文档中记录的指令适用于所有项目，提供了一致的操作基础。

4. **沉淀抽象规则 (`rules.md`)**： `rules.md` 用于记录项目中特定的抽象规则和设计模式。它包含了在对话中沉淀下来的关键原则和常用指令。
   - **动态沉淀**：在对话过程中，可以让 `Gemini CLI` 自行总结并记录到 `rules.md` 中。
   - **领域知识库**：为特定领域（如UI规范、API调用格式等）建立专门的规则文档。

- **按需引用**：在开始新任务时，让 `Gemini CLI` 先读取相关的规则文档，确保其行为符合预设的规范。

这是一种更高级的用法，其本质是**记录并复用抽象规则**。你可以让 `Gemini CLI` 把对话中沉淀下来的关键原则、设计模式或常用指令，保存到一个或多个专门的规则文档中。

- **动态沉淀**：在对话过程中，你可以让 `Gemini CLI` 自行总结。例如：“`Gemini CLI`，根据我们刚才的讨论，请将关于图片生成的几条核心规则，沉淀并追加到 `image_rules.md` 文件中。”
- **领域知识库**：你也可以为特定领域（如UI规范、API调用格式等）建立专门的规则文档。
- **按需引用**：在开始新任务时，让 `Gemini CLI` 先读取相关的规则文档，确保它的行为符合预设的规范。例如：“`Gemini CLI`，在开始编码前，请先学习 `api_rules.md` 中的所有规则。”
- **命名建议**：通常，项目相关的具体规则可放入 `rules.md`，而那些针对 `Gemini CLI` 行为本身的、更通用的全局指令，可以考虑放在 `gemini.md` 文件中，以便跨项目复用。

5. **进行AI辅助复盘**:

这是一种将日志价值最大化的好方法。在任务完成后，你可以让 AI 扮演“复盘教练”的角色，对整个开发过程进行总结和反思。

- **核心方法**: 利用 `dev_log.md` 作为输入，让 `Gemini CLI` 读取并分析整个过程。
- **示例指令**：“`Gemini CLI`，请通读 `dev_log.md`，为本次任务撰写一份复盘报告。报告需包含：1. 整体流程总结；2. 遇到的主要困难及解决方案；3. 三条可以提升未来效率的建议。”
- **好处**：将开发日志从单纯的过程记录，升华为提炼经验、优化未来工作流的宝贵资产。

5. 善用“计划模式”（Plan Mode）工作流

“计划模式”（Plan Mode）并非 `Gemini CLI` 的一个官方内置功能，而是一种我们强烈推荐的高效工作流，能极大提升 AI 处理复杂任务的准确性和可控性。其核心思想是：**先让 `Gemini CLI` 规划（Plan），再审查，最后执行（Execute）**。

如何实践?

1. **下达规划指令**: 当面对一个复杂需求时，不要直接让 `Gemini CLI` 开始工作。而是先要求它输出一个详细的、分步骤的行动计划。
   - **示例指令**: “ `Gemini CLI` ，针对‘批量生成素材’这个任务，请先给我一个详细的行动计划，列出你需要执行的每一步。暂时不要执行任何操作。”
2. **审查与调整**: 你会得到一个步骤列表。请仔细审查这个计划，如果发现有遗漏或不合理之处，可以和它讨论并调整，直到计划完善。
3. **授权执行**: 当你对计划满意后，再授权 `Gemini CLI` 开始执行。
   - **示例指令**: “ `Gemini CLI` ，这个计划很好，请现在开始执行。”

为什么这样做?

这种方式能确保 AI 的思考路径与你的预期一致，将复杂的黑盒任务分解为清晰、可控的子步骤，从而有效降低错误率。

6. Vibe Coding 核心原则

Vibe Coding 是一种与 AI 深度协同的工作哲学。为了帮助你更好地实践，我们总结了以下三个核心原则：

1. **信任AI的自主性，专注于高层抽象**:

   尽可能将具体工作授权给 `Gemini CLI` ，包括梳理需求、分析现状、对比差异，甚至主动提出改进建议。你的角色应更多地是提供最核心、最抽象的思路，并在关键节点（如审美、架构决策）进行引导和修正，而不是事必躬亲地指导每一步。

2. **善用外部记忆，保持上下文专注**:

   AI 的短期记忆是有限的。要学会通过文档（如 `rules.md` , `dev_log.md` ）来沉淀关键经验和过程。在需要 `Gemini CLI` 专注解决某个局部问题时，可以果断地清空（Clear）当前对话上下文，然后让它通过读取文档来获取所需的核心知识。这就像为它配备了一个可随时查阅的、永不遗忘的“第二大脑”。

3. **主导AI驱动的快速迭代**:

相信 `Gemini CLI` 在微观层面的迭代速度和能力超越人类。你的核心任务是**定义好目标**（例如一张完美的参考图片），并**主持好“迭代-测试-反馈”这个循环**。让 `Gemini CLI` 自主地、快速地朝目标迭代，识别问题，并列出改进项，然后继续迭代。你只需控制好这个反馈循环的节奏和最终目标，让 AI 成为自我优化的主角。

提交方式

请将包含源代码和心得文档的 GitHub 仓库链接发送给 **@大康Silicon**。

激励

对于通过验收标准的优秀候选人，我们将提供 **100元** 的现金激励，以感谢你投入的时间和精力。

祝你顺利！