

# 操作系统理念解析：网页索引的 OS 风格设计与模板实践指南

觉 空

## ABSTRACT

The Web Indexing OS-Style Template redefines webpage indexing by treating the index page as an entry-point operating system rather than a content display system. It prioritizes intuitive navigation, repository-based storage, and seamless file access through browser-native tab management. By delegating process and window management to the browser, this template maintains a minimalist, high-availability structure that ensures logical consistency. With features like automatic update reports, card-based layouts, and multi-faceted file stacking, it offers an efficient, self-contained solution for organizing and accessing project-related resources.

## 第一章 操作系统的概念与交互逻辑

操作系统这一概念自诞生以来便经历了不断的扩展与演变。从早期简单的资源调度工具，到如今复杂的人机交互平台，操作系统始终围绕如何更加直观、高效地管理系统资源和用户活动这一目标而发展。为了明确本项目所涉及的先进交互逻辑的具体含义，我们首先需要澄清操作系统的定义和其核心功能。

### 交互逻辑的基本概念

在本项目的背景下，我对操作系统的定义聚焦于其作为资源索引与管理入口的角色，而非内容本身的展示平台。具体而言，操作系统的核心作用在于维护文件的拓扑结构以及管理各类窗口和进程的生命周期，而不直接参与具体内容的呈现。这种设计理念强调了操作系统作为抽象管理层与实际资源（例如文件、设备或应用程序）之间的清晰边界。这一定位不仅符合模块化设计的原则，也提升了系统本身的灵活性和可扩展性。

在一个功能完整的操作系统中，通常需要考虑三类核心入口：系统管理型软件入口，如进程管理工具（例如Windows系统中的命令提示符CMD）；IO外设管理软件入口，如控制键盘、鼠标、相机、屏幕亮度与音量等外设以及网络连接的软件；以及文件入口，后者不仅包括文件本身，还包括为打开和编辑文件而设计的应用程序。这些应用程序虽然可能部分内置于系统中，但在功能定位上明显区别于前两类，它们并不参与对系统控制的核心操作。

## 交互与取用的分离

然而，当我们尝试将前两类入口（系统管理和外设管理软件）纳入以文件为中心的索引结构时，就会产生一些固有的问题。这些问题的根源在于交互型和取用型操作本质上的差异：取用型操作（如打开文件）是封闭且明确的，它们仅在当前的操作界面内起作用，并不会影响其他无关的资源或界面；而交互型操作（例如调整亮度、音量、网络状态）具有全局影响，往往不能局限于单个界面或上下文之内。

举例来说，智能手机屏幕边缘滑动打开的控制中心，就很好地体现了交互型入口的设计理念。这种入口直观地独立于文件系统之外，用户可以随时调用进行交互操作，不会与文件取用的逻辑产生混淆。反之，如果将屏幕亮度调整这一功能错误地置于文件索引结构中，用户在寻找文件时意外触发了亮度控制，将直接导致交互逻辑的混乱与用户体验的割裂。

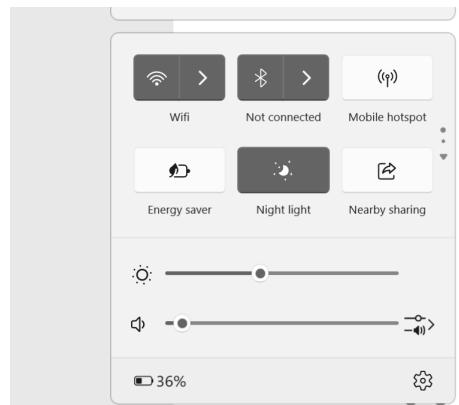
因此，我们所提出的先进交互逻辑正是基于对“交互”与“取用”两种操作的严格区分。在这一分类体系下，“交互”包括所有对设备和系统状态产生即时影响的操作，如相机拍摄（直接控制硬件）、浏览器访问网络数据（直接操控网络资源）以及遥控无人机的应用（实时控制外部设备）。而与之对应的“取用”则仅包括打开文件及其关联应用程序的操作，这类操作不产生全局性的状态影响，具有清晰的操作边界。

通过上述分析，我们可以清晰地认识到，智能设备中真正适合纳入文件结构管理的只有纯粹的文件及用于开启这些文件的应用程序，而交互型功能则应当独立设计为特殊的入口，从而维持文件结构的完整性与清晰度。这一设计原则不仅保证了系统内部结构的简洁与配置的简便性，更为用户提供了自洽且高效的交互体验，显著提高了操作系统在实际应用场景中的可用性与可靠性。

## App模式的设计思路解析

App模式通过将系统管理型软件以及大部分IO外设管理软件转化为可灵活配置的系统级组件，从而实现了通过全局控制中心的统一访问。这一策略既保证了交互方式的直观性，也清晰地将交互操作与文件访问操作作出明显区隔。一些次要的或备份性质的IO外设入口，以及负责文件打开任务的各类应用程序，则统一作为基础层级放置于桌面之上。此外，文件的整体拓扑索引结构被进一步抽象为单个专属的“文件”app，其自身被视作系统中独立的应用存在。

这种设计有效地从直觉层面区分了系统控制类软件与一般文件访问器，尤其在 app 分类时，管理 IO 外设的软件得以明确地区别于打开文件的软件，进一步提升了用户的交互效率。



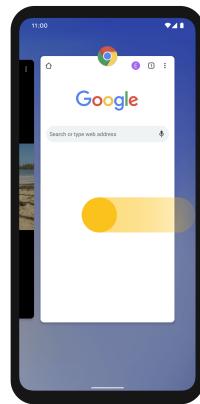
*Windows 11 系统中的控制中心（模拟塑料卡片材质）*

然而，这种模式存在显而易见的不足：当用户进入文件树后，需在复杂且庞大的路径结构中逐一检索各类文件，导致操作效率降低，用户难以迅速直达所需内容。同时，这种模式在需要频繁切换软件以处理同一文件的专业应用场景中表现欠佳。因此，我们提出进一步的优化方案：将原本单一的“文件”app 转变为多个独立的文件app，使之脱离传统文件系统中的统一根结点。每个文件或文件夹可同时拥有多个不同的路径进行索引，避免了在传统单一路径系统中产生的文件冗余。例如，一个名为“准则”的文件夹，可同时归属于多个不同项目而无需重复存储，用户能通过多条路径直达同一目标。这种模式尤其适合共享资源的多场景利用。

另外值得一提的是，这种情况下，直接作为系统一部分的“文件”入口必须支持多种打开方式，并允许用户自主选择窗口的开启模式（如多开或新建窗口），以确保交互的灵活性与操作效率。

## 构建更加符合操作直觉的系统逻辑

由上述思路继续逐步发展，我们得以明确勾勒出一个完整、契合用户直觉操作习惯的系统逻辑框架：我们延续了App模式的主要优点，将进程管理及 IO 控制组件等所有交互型软件集中整合至屏幕边缘或全局快捷入口（如屏幕边缘滑出或快捷手势），从而消除传统系统界面中侧边栏和页脚中散乱而多余的人口或工具。同时，我们进一步摒弃了 app 桌面入口对这些交互软件的备份，使桌面界面仅专注于承载“取用型”的文件及与文件深度绑定的应用程序（例如游戏或记事本等）。用户在打开文件夹时，可便捷地选择所需的打开方式：轻触即可默认调用预设的打开方式，长按则可快速弹出并选择其他可用的打开选项。此外，文件的打开动作默认通过新窗口或小窗模式展现，使操作逻辑更为贴合用户的直觉，进一步提升了整体使用体验的流畅性与便捷性。



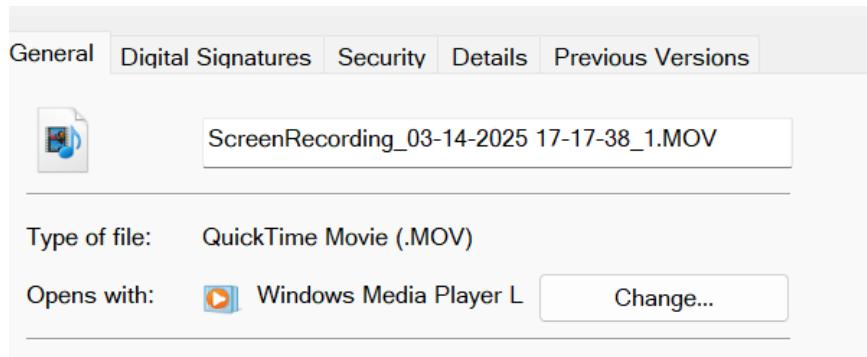
*Android 中边缘触控作为一种全局快捷入口*

### **需要提及的特殊软件：Terminal**

在上述的逻辑阐释中，Terminal 是一种值得特别说明的软件工具。正如我们此前曾提到的，Terminal 本质上是一种资源管理器，通过嵌入命令实现代码的执行，从而为那些原本不具备图形化界面的软件提供了直观易用的交互界面。这些软件中便包括一些文件打开类型的应用。在实际的文件打开操作过程中，我们无法直接选取这些缺乏独立界面的文件打开器；此时，我们需要将特定的文件打开器与 Terminal（cmd）组合起来使用，以便顺畅地访问文件内容并进行后续处理。

### **另外一种特殊的软件工具：配置器**

除此之外，我们还需要特别指出另一种重要的软件工具，即配置器。配置器涵盖了两类主要的配置任务：针对特定文件的专属配置与非特定文件的通用配置。这种配置既包括默认打开方式的确定，也涉及其他特别的参数设定。以往的操作系统中，“配置”这一功能常以文件属性（Properties）工具的形式存在。而在我们设计的系统中，我们更进一步地将配置器视作一种特殊的编辑器，由此使其能在文件结构内以更加直观、更加流畅的方式进行调用和管理。



*Windows 系统中的属性界面*

需要注意的是，诸如复制和移动等操作在我们的逻辑体系中并不归属于“编辑”的范畴，而更近似于一种交互型的操作方式。这类操作应当通过统

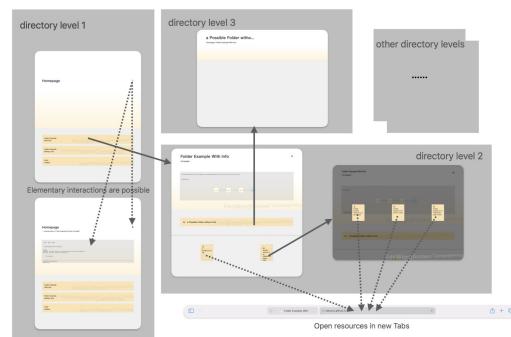
一的控制中心或者快捷操作激活特定的编辑界面模式进行。

## 批量处理

最后，针对文件的批量操作，我们设计了一种更为便捷的处理方式。若查看器本身即支持批量打开功能（例如配置器等），则我们允许用户直接通过长按背景区域来进入文件的多选状态；长按文件夹则默认快速地选中该文件夹下全部的文件。值得一提的是，这种长按手势同样属于进程管理类的快捷交互范畴，其目标在于进一步提高操作效率与系统响应的直观性。

## 第二章：基于浏览器实现的简化交互逻辑预览

在前述系统逻辑的基础上，我们还可以进一步抽离出仅用于查阅和预览的场景，在既有的操作系统中，利用浏览器的基本功能做出更加轻量化、易实现的简化版本用于体验。在网页实现的语境下，我们仅需考虑一种文件类型——即URL形式的文件链接。这一设计不仅天然排除了批量打开文件的复杂情况，还使原本需要系统处理的进程与文件窗口管理彻底简化为浏览器的“打开新标签页”操作。这意味着索引网页以一种依附于浏览器的操作系统视角，纯粹作为文件与资源的索引入口存在，并借助浏览器与仓库平台（如GitHub）完成数据存储与内容展示的跳转工作，从而实现高度精简的管理模式。得益于这样的设计，我们能够构建出配置简单、高度可用且具备自治跳转逻辑的网页模板。



示例网页的结构示例 <https://kakukuu.github.io/Web-Indexing-OS-style-Template/index.html>

## 第三章：网页模板的直觉感构建及其两个核心层次

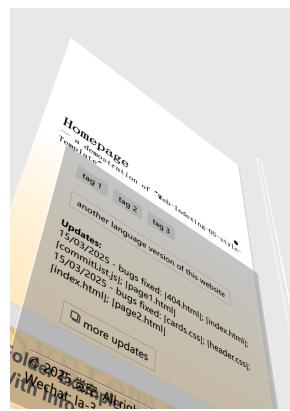
本项目提供的不仅仅是对概念性操作系统逻辑的简化实现，同时更可直接作为个人主页或项目资源发布平台的实用模板。我们明确摒弃了直接展示具体

内容的传统做法，而是将设计重心完全放在“索引”与“简介”这两个核心功能上。通过直观且符合操作直觉的卡片式布局、高效而精炼的跳转逻辑，以及贯穿页面始终的上下文提示，本模板为用户创造了一种统一而协调的交互体验。此外，为进一步提升模板的实用价值，本项目还整合了基于 GitHub Pages 的自动更新报告功能，以最小的维护成本实现信息动态同步。

下文将基于实际代码进一步详细阐明模板的具体设计思路、架构实现及其实际应用方式。

## 平面操作直觉与无结构视觉元素的融合

屏幕作为信息展示和交互的载体，最契合的是以二维空间为基础的直觉操作。这种直觉促使我们在设计时优先选择具备明显平面属性的视觉元素，以便用户能够直观、准确地感知并与之交互。在这个基础之上，我们所选择的视觉元素不仅应具备明确的平面属性，更宜摒弃内在复杂的拓扑结构，以避免与文件系统的交互产生混淆。因此，本模板在视觉元素的选取上倾向于天然缺乏结构的自然事物，例如晴朗无云的天空、弥漫的轻纱薄雾等。这些自然事物本身没有清晰界定的内部层级关系，可在直观上作为平面化的背景或辅助视觉元素使用，从而在不干扰文件和进程交互直觉的情况下，建立起视觉上的平静与秩序感。同时，这种视觉上的平静也有助于引导用户将注意力更多地集中于内容交互本身。



平面化的各个层次

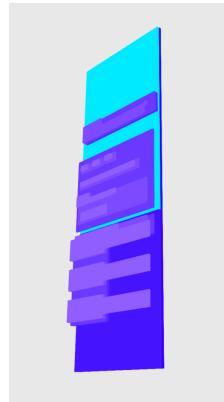
## 平面操作直觉与视觉元素的结构选择

屏幕作为信息交互的载体，本质上契合用户对二维空间的直觉性理解。因此，我们在模板设计之初便优先考虑平面属性明确的视觉元素，以便用户更直观、精准地感知界面布局并进行交互。为了进一步强化这种交互上的直觉性，我们在视觉元素的选取中尤其避免了那些内部拓扑结构复杂的对象，以防止与文件系统交互逻辑相混淆。在此基础上，我们倾向于选取具备明显平面感的材质，比如卡片、塑料片或毛玻璃，并搭配无云的天空或薄雾等天然缺乏明确结构的自然色彩元素，藉此实现视觉上的平静与秩序。这种设计策略有效地引导

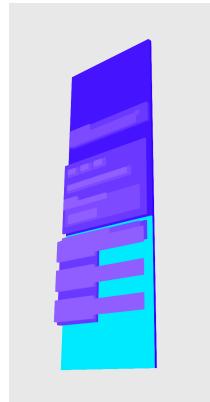
用户注意力集中于交互本身，而非视觉元素自身，从而在交互直觉与视觉感知之间达成平衡。

## 交互部（延展）与内容部（纯文件展示）的直观分层

在第一章中，我们探讨了文件访问与交互逻辑的分离。然而，在文件索引过程中，仍然存在某些必须依赖交互操作的场景，例如批量打开文件。这表明，在尽可能精简交互元素的同时，仍需为必要的交互操作预留合理的空间。因此，本模板在设计上进一步明确了功能分层的策略，将屏幕布局划分为两个主要区域：下半部分的“内容部”专注于文件的展示与索引，确保信息呈现的纯粹性，而上半部分的“交互部”则扩展至屏幕边缘，承担路径导航、文件描述以及各种辅助提示等功能，使交互组件集中管理，避免界面元素的分散冗余，从而在最大程度上优化用户体验和操作直觉。



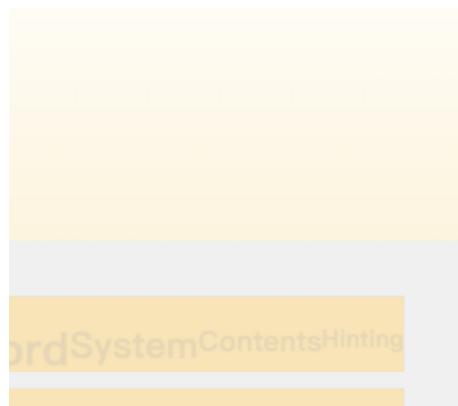
交互部



内容部

为清晰地区分这两个功能区域，我们采用了「张满」的视觉布局，即将交互部水平铺展至屏幕左右边缘，形成视觉上的无边界感与延伸感。这种布局不仅自然强化了上下区域的差异，更巧妙地将用户的视觉焦点引导至屏幕顶部，从而便捷地获取必要的路径信息与文件概要。此外，交互部同时作为内容区域的最后提示点，文件夹标题下方的“Info Block”可通过文字描述或结构图的形式呈现，使用户无需深入浏览，即可直观地把握文件夹的内容与用途。

在色彩选择上，为了确保上下区域在视觉上的协调性与层次感，我们特别强调了颜色的连续性，使交互部与内容部既能自然融合，又能在功能上形成清晰区分。基于这一原则，在取材上我们选择了高湿度环境下的淡色黄昏天空图景，并以卡片元素进行叠加呈现。这种柔和且统一的配色不仅赋予界面平衡感，也进一步强化了文件元素与交互元素之间的关联性，使它们在同一视觉空间中既能彼此衔接，又保持功能上的独立。借助这一设计理念，文件与文件夹的排列仿佛是界面上自由散落的碎片，而交互部则更像是漂浮于其上的信息引导层，为用户提供清晰的操作指引。



## 材质特写

此外，在此基础上，我们可以进一步增强卡片内容的灵活性，以适应不同场景的需求。例如，可以为文件夹标题添加封面图，以提供更直观的视觉识别，或者在特定情况下省略部分 "info block"，减少信息冗余，使界面更加简洁清晰。

这种高度自由的卡片展示方式尤其适用于个人研究主页或资源发布型网站，在这些场景中，我们通常不会在首页直接展示全部文件，而是通过首屏的差异化设计突出研究内容或导航信息。在这种情况下，我们可以扩大交互部的面积，使其覆盖整个屏幕，同时充当项目封面，以增强页面的层次感和引导性。这种设计不仅优化了索引结构，使其更加清晰，同时也使用户的交互过程更加直观、顺畅。通过这种方式，交互部不仅仅是一个辅助信息层，而是页面信息架构中的关键引导元素，引导用户快速理解页面结构，并在必要时提供直观的路径指引。

作为对上述交互设计的进一步拓展，这种布局模式在适当扩展后仍然具备较强的兼容性。例如，我们可以在交互部引入多个高度耦合的交互元素，使其形成一个功能丰富的控制区域，从而模糊传统意义上内容展示与交互控制的界限。这种设计允许用户在同一界面中完成多种操作，提高整体使用效率。

事实上，许多现代应用程序已经采用了类似的交互方式，例如网易云音乐（NetEase）在 iPad 端的界面布局，便是多交互部高度耦合的典型案例。其界面设计融合了内容展示与交互操作，使用户可以在同一屏幕上完成播放控制、歌单浏览和其他操作，而无需频繁切换页面。

然而，尽管这种设计在某些场景下极具优势，我们仍需权衡其适用性。在强调简洁性和直觉操作的文件管理系统中，过于复杂的交互区域可能会破坏清晰的层级结构，使用户难以快速定位和操作。因此，在我们的设计理念中，这种多层次交互的应用应当有所节制，以确保界面既具备足够的灵活性，又不失简洁直观的操作逻辑。

## 第四章：更多细节的实现

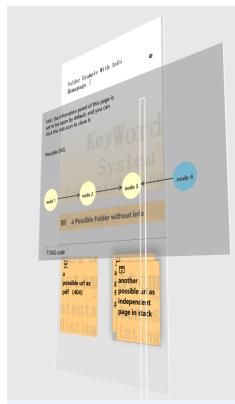
### 文件堆叠（Stack）模式与背景的动态变化

为了更加直观且高效地处理具有相同名称但具备不同功能或内容差异的文件，我们在界面中引入了堆叠（Stack）的概念。这种堆叠模式将具有强关联性的多个文件或功能聚合为视觉上统一、操作上便捷的卡片堆栈。用户在初始状态下只会看到单一卡片的顶部视图，既避免了界面杂乱的情况（图 1），又有效传递了这些文件在逻辑和功能上的关联性。点击或展开堆叠后，用户可便捷地访问每个文件或功能，并根据需要灵活切换。

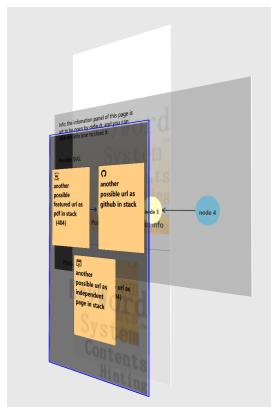
Shapley.aux	24/01/2025 19:54
Shapley.fdb_latexmk	24/01/2025 19:54
Shapley.flst	24/01/2025 19:54
Shapley.log	24/01/2025 19:54
Shapley.pdf	24/01/2025 19:54
Shapley.synctex.gz	24/01/2025 19:54
Shapley.tex	24/01/2025 19:54

一次编译中，*latex* 产生的大量不同后缀的文件

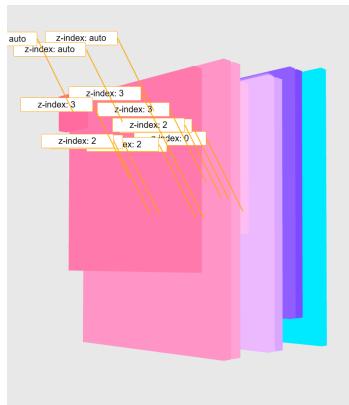
在这种堆叠交互的过程中，我们引入了显示层级与颜色的动态变化，以增强界面的直观性。比如当用户对堆叠文件组进行操作时，“stack”的图层位置（z-index）将从卡片背景的 0 变为高于所有其他元素的 6，突出当前堆叠文件组的焦点地位，同时逐渐变暗，以避免视觉上的干扰并进一步优化用户的交互体验。



收起状态的 Stack



展开状态的 Stack



Stack 中的多个卡片在收起状态下交错堆叠

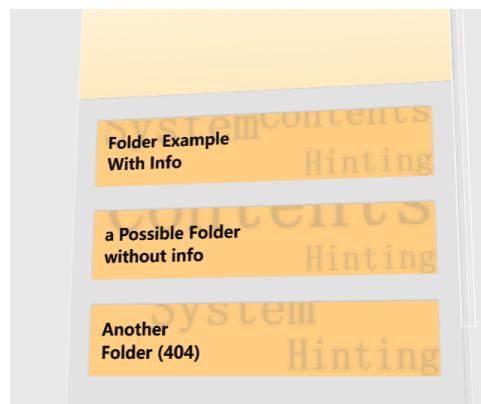
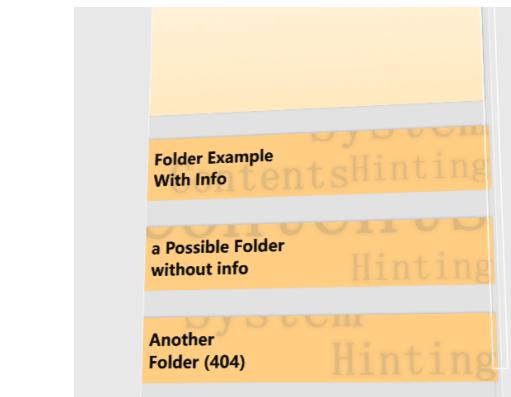
## 模板元素的精简与直觉化优化策略

### 卡片宽度的自适应布局方式

在本模板的界面设计中，我们摒弃了固定的宽度设置，而是根据屏幕本身所具有的平面直觉，赋予每个卡片元素一种相对宽度的属性。

具体而言，按照三种不同的屏幕尺寸等级，将卡片元素的宽度设置为屏幕宽度的某一百分比。这种设计模式的优势在于，无论页面显示于何种尺寸的屏幕上，元素的排版效果都能维持始终如一的稳定。同时由于网页的整体宽度总是与屏幕宽度相匹配，也避免了横向滚动条的出现，从而减少了操作的干扰。

尤其是考虑到对外层操作系统的手势与导航方式的兼容性。如果左右滑动用于前后导航，上下边缘滑动专用于多任务管理。那么这些手势与界面元素的交互互不干扰，确保了操作的流畅与操作体验的完整。



宽度小于 480px 时，文件夹卡片的宽度几乎占满屏幕

## 卡片关键词背景与内容提示逻辑

为了在视觉上强化文件卡片的信息提示功能，本模板进一步引入了卡片背景关键词系统。这种系统允许在卡片的背景图层上以半透明或浅色的方式展示关键字或标签内容，用户无需显式进入卡片即可直观了解文件的大致属性或内容类型。这种设计显著降低了认知负担，有效提升了文件的检索效率，同时也赋予界面更为丰富和多层次的视觉提示。

该关键词系统可以应用在卡片，也可以应用在页首的交互部，在处理文件夹时，通过匹配两者达到视觉上的连贯体验。（如图）



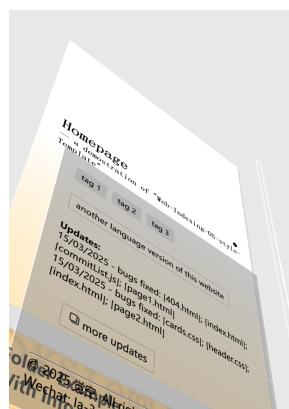
## 关键词系统在交互部的应用效果

### 精简传统网页元素，优化交互入口

在以往的网页设计中，页脚和侧边栏通常充斥着大量零散的工具和入口，这不仅增加了界面复杂度，也与我们强调的交互逻辑相悖。因此，本模板完全移除了传统的页脚以及侧边栏导航中散落的工具和入口，并将所有关键的链接或工具入口统一安置在页面顶端的信息区域（info区块）中，使整体导航结构更为集中、简洁且一目了然。信息区域以简洁的文本链接形式呈现，确保了在界面直觉上的统一性。

### 版本选择

以前的做法是将网站版本的选择（如语言和地区）放置在页面的其他区域，可能是页脚或侧边栏，通常以分散的链接或下拉菜单形式呈现。这种设计虽然功能上可以实现切换，但存在分散和复杂的问题。尤其是侧边栏或页脚中可能充斥着其他无关的工具和链接，增加了界面的复杂度。这导致用户需要额外的认知成本来理解这些选项的功能和位置，尤其是在页面设计不统一的情况下。相比之下，当前的做法将网站版本选择集中放置于页面顶部的 info 区域，通过带有链接跳转标签呈现，并在同一标签页内切换。甚至可以进一步引入地图作为交互索引，直观明了。避免了侧边栏或页脚的冗余设计，使页面更加简洁和高效。



## 网页跳转

跳转链接在传统网页设计中往往被归为杂元素，主要原因在于其缺乏统一的设计规范和清晰的视觉呈现。它们通常以零散的形式分布在页面的不同区域，例如侧边栏、页脚或正文中，既不直观也不易识别。这种设计导致用户在寻找跳转链接时需要额外的认知成本，甚至可能忽略其存在。此外，跳转链接的样式和行为往往与页面其他元素不一致，容易破坏整体的视觉和交互逻辑，进一步加剧了用户体验的割裂感。



关键词系统在不同卡片中的应用效果

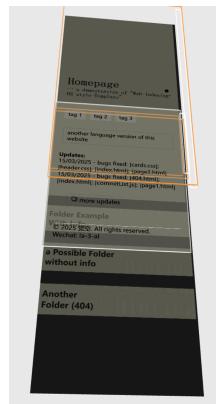
## 导航

传统的侧边栏导航在网页设计中长期占据重要地位，但随着用户需求的变化，其弊端也逐渐显现。侧边栏通常承载了大量的导航链接和工具入口，这种设计虽然功能齐全，却容易导致界面层次复杂化。

为了解决这些问题，我们提出了“干掉侧边栏导航”的设计理念，通过完全结构化的路径显示取而代之。所有的导航层级被清晰地呈现在页面顶部的交互部中，用户可以一目了然地同时了解当前所在的位置以及其他索引路径。这种设计不仅简化了界面布局，还显著提升了导航对不同文件结构的兼容性。

## 设置

值得特别提及的是，本模板还彻底取消了传统的“设置”选项，这一点最为典型的体现便是显示模式的自适应功能。我们通过检测用户设备或系统所采用的日间或夜间模式，自适应地调整网页界面的颜色主题，彻底消除了用户手动切换显示模式的繁琐操作。



黑夜模式的显示效果

## 其他

此外，如果外部浏览器和操作系统能够支持以下特性：文件在被打开时优先以小窗口模式进行预览，只有当用户明确需要更深入的访问时，才通过浏览器打开新标签页以完整显示；那么将进一步提升用户体验。通过这种方式，用户可以在不同场景下灵活切换操作模式，既满足了快速预览的需求，又兼顾了深入访问的便利性。



Flyme-OS 中实现的小窗打开逻辑

### 3.3 字体选择与视觉体验的优化

字体排版作为一项优雅的技能，在信息时代已被逐渐荒废。然而，本模板专门实现了不同内容处衬线体与无衬线体字体的区分，并通过字重的变化优化视觉效果。此外，模板使用了开源的思源宋体和思源黑体，通过 CDN 在线加载显示，确保了字体的美观与一致性。

## 第五章：模板使用说明

以下的模板的使用说明，分别介绍了目录结构、主要文件的作用，以及如何依据需求进行自定义与扩展。

### 项目文件

在整体项目结构中，所有文件与资源都需与示例保持相同的层级与路径，以确保相关链接和脚本引用能正常工作。HTML 文件（如 `index.html`、`page1.html`、`page2.html` 等）负责页面的主体结构与内容排布；`assets/images` 文件夹存放各种图片与图标资源，若需更换其中素材只需将新文件放入此目录并在 HTML 或 CSS 中更新引用路径。`css` 文件夹中包含了 `reset.css`、`styles.css` 和 `responsive.css`，这些样式文件分别用于清除默认样式、定义基础布局与视觉风格，以及处理移动端或大屏环境的适配。对于各种交互逻辑与页面动态效果，`javascripts` 文件夹中的 `infoToggle.js`、`commitList.js`、`copyCommitList.js`、`scrollHandler.js`、`throttle.js` 和 `main.js` 分别控制信息面板展开、更新列表读取与复制、滚动优化及初始化核心交互功能等。如果您不需要特定功能，可选择性移除对应脚本，并删去相应的 HTML 引用。

### 交互部

在首页示例（`index.html`）的设计上，交互部区域通过 `index-header` 与 `index-header-content` 类名确定布局，可直接添加标题与签名/路径。此处的信息面板（`infoPanel`）默认是收起的，点击右侧的图标可展开更多更新内容或跳转链接。一些场景中，您可以在信息面板里放置如“多语言版本”等实用的可点击标签，也可利用 JavaScript 脚本，如 `commitList.js` 动态获取项目更新信息并展示在此处；同时也可以一键复制的便捷操作，如 `copyCommitList.js`。

在普通页面（如 `page1.html`、`page2.html` 等）中类似，但交互部并不采用与首页相同的类名，而是以 `page-header` 或 `page-`

`header-content` 进行排版区分，让用户可直观感受到这是项目中的“子页面”或“附属页面”。

如果某个文件夹的整体有一些关键词，那么关键词背景（`keywords-background` 与 `keywords` 容器）也可以应用在页首的部分，既能带来视觉连续性，也能暗示页面主题，它由多个关键词 `<span>` 组成，通过不同字号来强化提示信息的权重。

需要特别注意的是，信息面板的展开与折叠是通过添加或移除 `open` 类名实现的，并由 `infoToggle.js` 脚本进行触发与控制。对于更高级的多层交互，您可以在同一页面的顶层或信息面板中引入更多脚本，也可整合第三方服务如 Google Analytics、Matomo 等，用于统计分析或其他功能扩展。

## 内容部

在交互部的下方，页面的内容部会以小卡片的形式展示各类文件夹或链接卡片，将视觉焦点集中在导航与核心资源上。

每个文件夹卡片的核心结构由 `<a>` 标签构成，链接至相应的文件夹页面。文件夹卡片可选通过 `` 显示文件夹图标，结合其自适应的长条形状，使用户直观理解其作用。

与文件夹类似，文件卡片呈矩形，用于展示URL索引的具体的文档或外部资源，如 PDF、GitHub 页面或在线独立页面。这些文件卡片的自动排布由 `<div class="card-container lecture">` 约束。同样，每个 `<a>` 标签链接至具体资源，由标签的链接属性 `target="_blank"` 决定在新标签页打开。

在某些情况下，多个文件可能具有相同的名称但内容不同，比如不同版本的文档或格式各异的资源。为了解决这种问题，模板提供了文件堆叠（Stack）功能。在 `<div class="card lecture stack">` 容器内，多种相关文件被组合在一起，以图标和后缀作为区分，使用户可以快速切换查看不同版本。每个文件仍然保持独立的点击入口，但所有文件在页面上表现为同一个卡片组。这种设计的一个典型应用是 GitHub 代码仓库与在线项目文档的结合，例如其中一个堆叠卡片链接至 GitHub 页面（``），另一个链接至项目的独立介绍页面（``）。在 `cardStack.js` 的控制下，用户可以更直观地展开或收起堆叠内容，从而提升页面的组织性与可读性。

```
<div class="card-container lecture">
```

我们通过控制 *card container* 的内间距来实现不同屏幕宽度下卡片距离屏幕边缘的距离，所以卡片需要放置在 *card container* 元素内部

```
<a href="404.html" target="_blank" class="card lecture">
  <h2 class="card-title lecture">
    <br>a<br>possible url as pdf [404]
    <!-- 外部引用：重复使用的 PDF 图标 -->
  </h2>
  <div class="keywords-background">
    <div class="keywords">
      <span style="font-size: 68px;">Key</span>
      <span style="font-size: 80px;">Word</span>
      <span style="font-size: 64px;">System</span>/br
      <span style="font-size: 50px;">Contents</span>
      <span style="font-size: 42px;">Hinting</span>
    </div>
  </div>
</a>
```

普通的卡片，直接放置在 *card-container lecture* 中

```
<div class="card lecture stack" href="404.html">
  <a href="#" target="_blank" class="card lecture subcard">
    <h2 class="card-title lecture"><br>another<br>possible featured url as pdf in stack [404]</h2>
    <div class="keywords-background">
      <div class="keywords">
        <span style="font-size: 68px;">Key</span>
        <span style="font-size: 80px;">Word</span>
        <span style="font-size: 64px;">System</span>/br
        <span style="font-size: 50px;">Contents</span>
        <span style="font-size: 42px;">Hinting</span>
      </div>
    </div>
  </a>
  <a href="https://github.com/kakaku/We-Indexing-OS-style-Template" target="_blank" class="card lecture subcard">
    <h2 class="card-title lecture"><br>another<br>possible url as github in stack</h2>
    <div class="keywords-background">
      <div class="keywords">
        <span style="font-size: 68px;">Key</span>
        <span style="font-size: 80px;">Word</span>
        <span style="font-size: 64px;">System</span>/br
        <span style="font-size: 50px;">Contents</span>
        <span style="font-size: 42px;">Hinting</span>
      </div>
    </div>
  </a>
  <a href="https://kakaku.github.io/We-Indexing-OS-style-Template-Project-Introduction/" target="_blank" class="card lecture subcard">
    <h2 class="card-title lecture"><br>another<br>possible url as independent page in stack</h2>
    <div class="keywords-background">
      <div class="keywords">
        <span style="font-size: 68px;">Key</span>
        <span style="font-size: 80px;">Word</span>
        <span style="font-size: 64px;">System</span>/br
        <span style="font-size: 50px;">Contents</span>
        <span style="font-size: 42px;">Hinting</span>
      </div>
    </div>
  </a>
</div>
```

*stack* 的层级处在 *card-container lecture* 与卡片之间，在堆叠卡片时，重复的关键词背景也可以移至 “*stack*” 层级以避免代码重复。

此外，所有卡片都支持和交互部相同的关键词背景系统（`<div class="keywords-background">`），使用户能够快速预览内容并理解其层次结构。

## 其他

关于自定义部分，无论是添加更多文件夹与卡片，还是更换视觉主题和背景，只需在对应的 HTML、CSS 或脚本文件中修改即可。如果需要多语言支持，可简单通过 ai 辅助，复制修改各个页面的html代码，并通过主页info中的超链接切换主页。

使用此模板时，通常只需将文件放置于任意静态网页服务器环境中（如 GitHub Pages、Nginx 或 Apache），然后在浏览器中访问相应的 HTML 页面即可预览。若文件路径或脚本逻辑出现错误，页面中的图标、脚本或外部链接可能无法正常加载，应及时检查相关引用是否匹配实际文件位置。完成这一步

后，您就可以根据项目需求做更深入的个性化编辑，比如新增页面、增加卡片、扩展 JS 功能，甚至移除部分并不需要的模块。

## Performance Comparison Test

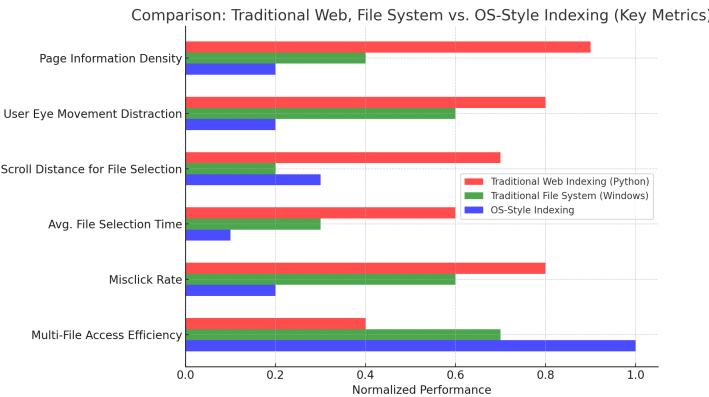
To validate the advantages of OS-Style Indexing, a simple comparison test can be conducted. The test involves performing the following tasks across different indexing methods:

**File Selection Time Test:** Open five randomly assigned files from a project using traditional indexing (e.g., README navigation, file explorer) and OS-Style Indexing. Record the total time taken.

**Misclick Rate Measurement:** Count the number of accidental misclicks when selecting files in each indexing system.

**Multi-File Access Efficiency:** Open and switch between five different files, measuring how smoothly navigation occurs without disrupting workflow.

**Information Density Perception:** Observe and note down how much additional text, menus, or irrelevant elements appear before finding the required files.



*By comparing these results, we can quantitatively assess how OS-Style Indexing improves efficiency and reduces distractions compared to traditional file access methods.*