# Chrome 内网智能化助手 - 需求文档 (V4.2 for AI Programming)

# 项目名称： Chrome 内网智能化助手

# 目标： 开发一个高兼容性、高稳定性、独立于外部软件的Chrome扩展程序。专注于内网环境，提供便捷的用户操作录制功能，支持灵活的回放模式（后台静默或前台展示），具备多任务管理、快捷键快速触发能力。在用户配合浏览器设置的前提下，实现文件自动下载到预设的本地文件夹并自动确认保存。同时，增强URL配置功能，支持通配符匹配和动态参数化。新增自动点击上传文件按钮功能，但文件选择仍需用户手动完成。

# 1. 核心功能描述

# 1.1 多任务脚本管理

# 功能： 用户可以创建、命名、编辑、删除、复制、导入、导出（JSON格式）自动化脚本。

# 脚本属性： 每个脚本包含名称、描述、创建时间、最后运行时间、状态（启用/禁用）。

# 脚本编辑： 提供图形化界面编辑脚本步骤，包括增删改查、调整顺序、修改参数。

# 1.2 增强型启动URL配置

# 功能： 每个脚本可配置一个起始URL，脚本执行时首先导航到此URL。

# 通配符支持： 启动URL可包含通配符（如 \*），以匹配URL中可变的部分。

# 示例： https://internal.example.com/item/\*/details

# 动态变量支持： 启动URL可包含占位符变量（如 {projectId}），这些变量可在执行前由用户输入或通过其他脚本逻辑提供。

# 示例： https://internal.example.com/project/{projectId}/dashboard

# 内置宏： 支持预定义宏，如 {DATE\_TODAY} (YYYY-MM-DD)，{TIME\_HHMM} (HHMM)，{USER\_INPUT\_VAR\_NAME} (触发用户输入)。

# 回放策略：

# 精确导航： 严格导航到解析后的完整URL。

# 通配符导航： 导航到第一个匹配通配符模式的页面。

# 录制时URL捕获： 录制器在开始时捕获当前URL，并提供用户编辑为模板URL的选项。

# 1.3 录制模块

# 功能： 录制用户在网页上的操作，并生成可编辑的自动化脚本步骤。

# 操作捕获类型：

# 点击事件： 记录用户对任何可点击元素的点击（按钮、链接、复选框、单选按钮、下载链接）。

# 文本输入： 记录用户在输入框中输入的文本。

# 下拉菜单选择： 记录用户在 select 元素中选择的选项。

# 页面导航/加载： 自动记录页面跳转，并确保回放时等待页面加载完成。

# 页面滚动： 记录滚动窗体的操作。

# 新增 文件上传按钮点击：

# 录制： 捕获用户对 <input type="file"> 元素或其代理元素的点击，将其标记为“上传文件按钮点击”步骤。

# 回放： 模拟点击此元素，触发浏览器文件选择对话框。用户必须手动选择文件。 脚本会等待用户完成选择后继续。

# 录制反馈： 录制时高亮当前操作的元素。

# 脚本编辑： 录制结束后提供界面编辑步骤，包括：

# 修改元素选择器（默认生成XPath，用户可编辑）。

# 调整步骤顺序。

# 删除、插入新步骤。

# 为每一步添加自定义等待时间。

# 修改输入文本或选择值。

# 配置下载文件名重命名规则。

# 配置上传后等待策略（如等待特定元素出现）。

# 1.4 回放引擎

# 功能： 按序执行脚本中的操作步骤，支持两种模式。

# A. 后台静默回放模式：

# 在不可见的Chrome标签页中执行，无用户界面显示。

# 无用户交互，任务完成后自动关闭标签页。

# 适用于定时任务、批处理。

# B. 前台展示回放模式：

# 在用户当前可见的Chrome窗口或新标签页中执行。

# 可视化执行，用户可看到每一步操作。

# 提供“暂停”、“继续”、“停止”控制按钮。

# 实时高亮当前操作元素。

# 适用于演示、调试、半自动流程。

# 共享回放特性：

# 多标签页/窗口支持： 能够在新的可见或不可见标签页/窗口中执行。

# 鲁棒元素定位： 优先ID，次之Name，然后Data属性，Class，最终使用XPath。失败时尝试备用选择器和重试。

# 动态等待：

# 固定延迟。

# 等待元素出现、可见或特定内容/属性值。

# 页面完全加载等待。

# 自定义等待步骤。

# 错误处理： 记录错误日志，支持重试、跳过步骤、停止任务。

# 会话管理： 利用当前Chrome会话Cookie保持登录状态。

# 1.5 受限文件下载管理

# 功能： 自动化下载文件，并提供文件命名建议。

# 核心限制：文件下载的自动保存到指定目录功能，要求用户必须手动在 Chrome 浏览器设置中：\*\*

# 关闭“下载前询问每个文件的保存位置”选项。

# 设置固定的“下载位置”为目标文件夹。

# 如果用户未进行上述设置，扩展程序无法实现自动保存到预设目录。

# 下载识别： 录制器识别触发下载的点击事件或链接。回放时，使用 chrome.downloads.onCreated 监听下载事件。

# 文件重命名与子目录建议：

# 利用 chrome.downloads.onDeterminingFilename API，根据脚本配置实时修改下载文件的文件名，并可在用户默认下载目录下创建子目录。

# 支持基于页面内容、日期/时间戳、脚本名称的复杂命名规则。

# 支持自动处理文件名冲突。

# 下载状态： 在任务日志中记录下载进度和完成状态（文件名、路径、状态）。

# 1.6 定时任务管理

# 功能： 为脚本配置定时自动执行计划（仅支持后台静默模式）。

# 配置项： 运行时间（时、分）、频率（每天、每周、每月、仅一次）、启用/禁用状态。

# 管理： 任务队列、优先级设置、运行历史记录。

# 1.7 快捷键触发执行

# 功能： 用户可为每个脚本分配一个全局或Chrome特定快捷键组合。

# 即时执行： 按下快捷键立即启动脚本，可选择后台静默或前台展示模式。

# 管理： 提供快捷键的添加、修改、删除。

# 1.8 用户界面 (UI)

# 扩展弹出窗口 (Popup)：

# 录制/停止按钮。

# 显示最近任务状态。

# 进入完整管理页面的链接。

# 显示当前页面可用的快捷键任务。

# “一键执行默认任务”入口。

# 独立管理页面 (Options Page)：

# 脚本管理： 列表、CRUD、导入/导出。

# 定时任务配置： 设定和管理定时任务。

# 快捷键配置： 绑定/解绑/修改快捷键与脚本，选择回放模式。

# 下载设置： 提示 Chrome 下载设置要求，配置重命名规则。

# 任务日志： 详细记录所有任务的执行结果、运行模式、耗时、错误信息、下载文件信息。支持过滤、导出。

# 全局设置： 默认延迟、超时时间、错误处理策略、日志保留。

# 2. 技术栈及平台

# 平台： Google Chrome Web Browser (桌面版)。

# 技术： Chrome Extension (Manifest V3)。

# 核心API：

# chrome.storage (数据持久化)。

# chrome.tabs / chrome.scripting (页面操作、注入脚本)。

# chrome.alarms (定时任务)。

# chrome.commands (快捷键)。

# chrome.downloads (文件下载监听、命名建议)。

# chrome.runtime / chrome.messages (扩展内部通信)。

# UI 技术： HTML, CSS, JavaScript (可能搭配轻量级框架如 Vue/React 或纯JS)。

# Chrome 内网智能化助手 - 终极开发指南

版本: 4.2 (AI编程助手专用版)

日期: 2025年10月13日

目标: 基于V4.2需求文档，为AI编程助手提供一份无歧义、可直接用于代码生成的“施工蓝图”。

## 1. 核心理念与架构

### 1.1. 项目愿景

打造一款专为企业内网设计的、完全独立的Chrome扩展程序。通过提供极致简单易用的网页操作录制与回放功能，自动化处理重复性任务，从而解放生产力。

### 1.2. 核心原则

* **独立自治 (Self-Contained):** 零外部依赖。所有逻辑与数据均在扩展内部处理，保证内网环境的兼容性与数据安全。
* **用户友好 (User-Friendly):** 为非技术人员提供直观的图形化操作，同时为高级用户保留足够的灵活性。
* **稳定可靠 (Robust & Reliable):** 引擎必须能优雅地处理现代网页的复杂性，包括异步加载、动态内容和常见错误。
* **安全至上 (Security First):** 严格遵循Manifest V3规范，确保扩展自身的安全，不引入任何安全风险。

### 1.3. 系统架构

采用经典的**事件驱动架构**，以service-worker.js为中心枢纽，负责所有后台逻辑和状态管理。其他所有组件（Popup, Options, Content Scripts）都作为无状态的“视图”或“执行器”，通过定义好的消息协议与Service Worker通信。

## 2. 项目设置

### 2.1. 文件结构

/  
|-- manifest.json  
|-- service-worker.js  
|-- icons/  
| |-- icon16.png, icon48.png, icon128.png  
| |-- recording.png (用于录制状态)  
|-- popup/  
| |-- popup.html  
| |-- popup.js  
|-- options/  
| |-- options.html  
| |-- options.js  
|-- content/  
| |-- recorder.js  
| |-- player.js  
|-- utils/  
| |-- storage.js  
| |-- selector.js  
| |-- uuid.js  
| |-- url\_parser.js

### 2.2. manifest.json

{  
 "manifest\_version": 3,  
 "name": "Chrome 内网智能化助手",  
 "version": "1.0.0",  
 "description": "录制并回放网页操作，自动化处理内网任务。",  
 "permissions": [  
 "storage",  
 "tabs",  
 "scripting",  
 "alarms",  
 "downloads"  
 ],  
 "host\_permissions": [  
 "<all\_urls>"  
 ],  
 "background": {  
 "service\_worker": "service-worker.js"  
 },  
 "action": {  
 "default\_popup": "popup/popup.html",  
 "default\_icon": {  
 "16": "icons/icon16.png",  
 "48": "icons/icon48.png"  
 }  
 },  
 "options\_page": "options/options.html",  
 "commands": {  
 "run\_task\_1": {  
 "description": "运行快捷任务 1"  
 },  
 "run\_task\_2": {  
 "description": "运行快捷任务 2"  
 }  
 },  
 "icons": {  
 "16": "icons/icon16.png",  
 "48": "icons/icon48.png",  
 "128": "icons/icon128.png"  
 }  
}

## 3. 核心数据模型 (chrome.storage.local)

### 3.1. 数据结构定义 (TypeScript-like)

// 类型定义 (仅为清晰起见，实际为JS)  
  
interface Script {  
 id: string; // UUID  
 name: string;  
 description: string;  
 createdAt: string; // ISO 8601  
 lastRunAt?: string; // ISO 8601  
 status: 'enabled' | 'disabled';  
 startUrl: StartUrlConfig;  
 steps: Step[];  
}  
  
interface StartUrlConfig {  
 template: string; // 支持宏变量的模板URL, e.g., [https://example.com/](https://example.com/){projectId}?date={DATE\_TODAY}  
 playbackStrategy: 'exact' | 'wildcard';  
}  
  
interface Step {  
 id: string; // UUID  
 type: 'click' | 'type' | 'select' | 'wait' | 'scroll' | 'navigateTo' | 'uploadClick';  
 selector?: string; // CSS Selector or XPath  
 value?: string | number; // 输入值, 等待时间, 滚动像素, 选择的option value  
 renameRule?: string; // 适用于触发下载的click步骤  
 postUploadWait?: WaitCondition; // 适用于uploadClick步骤  
 waitType?: 'elementVisible' | 'time' | 'navigation';  
 timeout?: number; // 超时时间  
}  
  
interface WaitCondition {  
 type: 'elementVisible';  
 selector: string;  
 timeout: number;  
}  
  
interface RecordingState {  
 isRecording: boolean;  
 targetTabId: number | null;  
 currentScriptId: string | null;  
}  
  
interface PlaybackTask {  
 taskId: string; // UUID  
 tabId: number;  
 scriptId: string;  
 currentStepIndex: number;  
 status: 'running' | 'paused' | 'error' | 'completed' | 'waitingForUserInput';  
 errorInfo?: string;  
 resolvedVariables?: Record<string, string>; // 存储用户输入的动态变量  
}  
  
interface Schedule {  
 id: string; // UUID  
 scriptId: string;  
 name: string;  
 frequency: 'once' | 'daily' | 'weekly' | 'monthly';  
 time: string; // HH:MM  
 enabled: boolean;  
}  
  
interface LogEntry {  
 id: string; // UUID  
 scriptName: string;  
 startTime: string; // ISO 8601  
 endTime: string; // ISO 8601  
 durationMs: number;  
 status: 'success' | 'failed';  
 trigger: 'manual' | 'schedule' | 'shortcut';  
 mode: 'foreground' | 'background';  
 summary: string; // e.g., "任务成功完成" 或 "在步骤5处失败: 元素未找到"  
 downloadedFiles?: { name: string, path: string }[];  
}

## 4. Service Worker (service-worker.js) - 中央控制器

**核心原则：无状态。所有函数都应是纯粹的事件处理器，从storage读取状态，处理后将新状态写回storage。**

### 4.1. 完整代码实现

// service-worker.js - 扩展的中央控制器  
  
import { Storage } from './utils/storage.js';  
import { generateUUID } from './utils/uuid.js';  
import { resolveUrl } from './utils/url\_parser.js';  
  
// --- 生命周期事件 ---  
chrome.runtime.onInstalled.addListener(async () => {  
 // 初始化存储结构  
 await Storage.set({  
 scripts: [],  
 schedules: [],  
 shortcuts: {},  
 logs: [],  
 settings: {  
 defaultWaitTime: 500,  
 defaultTimeout: 30000,  
 errorHandlingPolicy: "stop", // 'stop', 'skip', 'retry'  
 logRetentionDays: 30  
 },  
 recordingState: {  
 isRecording: false,  
 targetTabId: null,  
 currentScriptId: null  
 },  
 playbackQueue: [],  
 });  
});  
  
chrome.runtime.onStartup.addListener(async () => {  
 // TODO: 重新注册所有在 storage 中持久化的 alarms  
 const {  
 schedules  
 } = await Storage.get('schedules');  
 if (schedules) {  
 schedules.forEach(schedule => {  
 if (schedule.enabled) {  
 // ... 注册 alarm 的逻辑 ...  
 }  
 });  
 }  
});  
  
// --- 监听器 ---  
chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {  
 const handler = messageHandlers[message.type];  
 if (handler) {  
 handler(message.payload, sender).then(sendResponse);  
 }  
 return true; // 保持消息通道开放以进行异步响应  
});  
  
chrome.commands.onCommand.addListener(handleCommand);  
chrome.alarms.onAlarm.addListener(handleAlarm);  
chrome.downloads.onDeterminingFilename.addListener((item, suggest) => {  
 handleDownloadRename(item, suggest);  
 return true; // 异步操作需要返回true  
});  
  
  
// --- 消息处理器 ---  
const messageHandlers = {  
 'GET\_STATE': handleGetState,  
 'START\_RECORDING': handleStartRecording,  
 'STOP\_RECORDING': handleStopRecording,  
 'RECORD\_ACTION': handleRecordAction,  
 'START\_PLAYBACK': handleStartPlayback,  
 'STEP\_COMPLETED': handleStepCompleted,  
 // ... 其他UI请求处理器  
};  
  
async function handleGetState() {  
 return await Storage.get(['recordingState', 'playbackQueue']);  
}  
  
async function handleStartRecording({  
 tabId,  
 url  
}) {  
 const newScriptId = generateUUID();  
 const newScript = {  
 id: newScriptId,  
 name: `新录制 ${new Date().toLocaleString()}`,  
 description: "",  
 createdAt: new Date().toISOString(),  
 status: 'enabled',  
 startUrl: {  
 template: url,  
 playbackStrategy: 'exact',  
 },  
 steps: []  
 };  
  
 const {  
 scripts  
 } = await Storage.get('scripts');  
 scripts.push(newScript);  
  
 await Storage.set({  
 scripts,  
 recordingState: {  
 isRecording: true,  
 targetTabId: tabId,  
 currentScriptId: newScriptId,  
 },  
 });  
  
 await chrome.scripting.executeScript({  
 target: {  
 tabId  
 },  
 files: ['utils/selector.js', 'content/recorder.js'],  
 });  
  
 await chrome.action.setIcon({  
 path: 'icons/recording.png',  
 tabId  
 });  
 await chrome.action.setBadgeText({  
 text: 'REC',  
 tabId  
 });  
}  
  
async function handleStopRecording() {  
 const {  
 recordingState  
 } = await Storage.get('recordingState');  
 if (!recordingState.isRecording) return;  
  
 await chrome.action.setIcon({  
 path: 'icons/icon48.png',  
 tabId: recordingState.targetTabId  
 });  
 await chrome.action.setBadgeText({  
 text: '',  
 tabId: recordingState.targetTabId  
 });  
  
 await Storage.set({  
 recordingState: {  
 isRecording: false,  
 targetTabId: null,  
 currentScriptId: null  
 }  
 });  
}  
  
  
async function handleRecordAction(action, sender) {  
 const {  
 recordingState,  
 scripts  
 } = await Storage.get(['recordingState', 'scripts']);  
 if (!recordingState.isRecording || recordingState.targetTabId !== sender.tab.id) return;  
  
 const script = scripts.find(s => s.id === recordingState.currentScriptId);  
 if (script) {  
 action.id = generateUUID();  
 script.steps.push(action);  
 await Storage.set({  
 scripts  
 });  
 }  
}  
  
async function handleStartPlayback({  
 scriptId,  
 mode  
}) {  
 const {  
 scripts,  
 playbackQueue  
 } = await Storage.get(['scripts', 'playbackQueue']);  
 const script = scripts.find(s => s.id === scriptId);  
 if (!script || script.status === 'disabled') {  
 console.error("脚本未找到或已禁用");  
 return;  
 }  
  
 // TODO: 实现处理动态变量输入的逻辑  
 const finalUrl = resolveUrl(script.startUrl.template);  
  
 const tab = await chrome.tabs.create({  
 url: finalUrl,  
 active: mode === 'foreground'  
 });  
  
 const newTask = {  
 taskId: generateUUID(),  
 tabId: tab.id,  
 scriptId,  
 currentStepIndex: 0,  
 status: 'running',  
 resolvedVariables: {},  
 };  
 playbackQueue.push(newTask);  
 await Storage.set({  
 playbackQueue  
 });  
  
 // 确保页面加载完成后再注入脚本  
 chrome.tabs.onUpdated.addListener(async function listener(tabId, info) {  
 if (tabId === tab.id && info.status === 'complete') {  
 await chrome.scripting.executeScript({  
 target: {  
 tabId: tab.id  
 },  
 files: ['content/player.js'],  
 });  
 executeNextStep(newTask);  
 chrome.tabs.onUpdated.removeListener(listener); // 清理监听器  
 }  
 });  
}  
  
async function executeNextStep(task) {  
 const {  
 scripts  
 } = await Storage.get('scripts');  
 const script = scripts.find(s => s.id === task.scriptId);  
  
 if (task.currentStepIndex >= script.steps.length) {  
 await completeTask(task);  
 return;  
 }  
  
 const step = script.steps[task.currentStepIndex];  
 chrome.tabs.sendMessage(task.tabId, {  
 type: 'EXECUTE\_STEP',  
 payload: {  
 step,  
 taskId: task.taskId  
 }  
 });  
}  
  
async function handleStepCompleted({  
 taskId,  
 result  
}) {  
 const {  
 playbackQueue  
 } = await Storage.get('playbackQueue');  
 const taskIndex = playbackQueue.findIndex(t => t.taskId === taskId);  
 if (taskIndex === -1) return;  
  
 const task = playbackQueue[taskIndex];  
  
 if (result.success) {  
 task.currentStepIndex++;  
 await Storage.set({  
 playbackQueue  
 });  
 executeNextStep(task);  
 } else {  
 task.status = 'error';  
 task.errorInfo = result.error;  
 await Storage.set({  
 playbackQueue  
 });  
 await logTaskCompletion(task, 'failed');  
 // TODO: 根据错误处理策略执行重试或跳过  
 }  
}  
  
async function completeTask(task) {  
 task.status = 'completed';  
 const {  
 playbackQueue  
 } = await Storage.get('playbackQueue');  
 const updatedQueue = playbackQueue.filter(t => t.taskId !== task.taskId);  
 await Storage.set({  
 playbackQueue: updatedQueue  
 });  
 await logTaskCompletion(task, 'success');  
 // 如果是后台任务，自动关闭标签页  
 const {  
 mode  
 } = (await chrome.storage.local.get('playbackQueue')).playbackQueue.find(p => p.taskId === task.taskId) || {};  
 if (mode === 'background') {  
 await chrome.tabs.remove(task.tabId);  
 }  
}  
  
  
// --- 其他处理器 ---  
function handleCommand(command) {  
 // TODO: 从 storage 读取快捷键配置并执行对应的脚本  
 console.log(`Command: ${command}`);  
}  
  
function handleAlarm(alarm) {  
 // TODO: 从 storage 读取定时任务配置并执行对应的脚本  
 console.log(`Alarm: ${alarm.name}`);  
}  
  
async function handleDownloadRename(downloadItem, suggest) {  
 const {  
 playbackQueue,  
 scripts  
 } = await Storage.get(['playbackQueue', 'scripts']);  
 const task = playbackQueue.find(t => t.tabId === downloadItem.tabId);  
 if (!task) return;  
  
 const script = scripts.find(s => s.id === task.scriptId);  
 // 注意：这里的 currentStepIndex 可能因为异步而超前，需要更精确的关联  
 const step = script.steps[task.currentStepIndex - 1]; // 假设下载是由上一步触发的  
  
 if (step && step.renameRule) {  
 const newFilename = resolveUrl(step.renameRule, task.resolvedVariables);  
 suggest({  
 filename: newFilename,  
 conflictAction: 'uniquify'  
 });  
 }  
}  
  
async function logTaskCompletion(task, status) {  
 // TODO: 实现详细的日志记录逻辑  
 console.log(`任务 ${task.taskId} ${status}.`);  
}

## 5. UI - 弹出窗口 (Popup)

### 5.1. popup/popup.html

<!DOCTYPE html>  
<html>  
<head>  
 <meta charset="UTF-8">  
 <title>内网助手</title>  
 <link rel="stylesheet" href="[https://cdn.tailwindcss.com](https://cdn.tailwindcss.com)"></link>  
 <style>  
 body { width: 300px; }  
 .btn-rec { background-color: #ef4444; }  
 .btn-rec.recording { background-color: #8b5cf6; }  
 </style>  
</head>  
<body class="bg-gray-100 p-4 font-sans">  
 <div class="flex flex-col items-center space-y-4">  
 <h1 class="text-xl font-bold text-gray-800">内网智能化助手</h1>  
  
 <button id="toggle-record-btn" class="w-full py-3 px-4 rounded-lg text-white font-semibold shadow-md transition-transform transform hover:scale-105 btn-rec">  
 开始录制  
 </button>  
  
 <div id="status-panel" class="w-full p-3 bg-white rounded-lg shadow">  
 <h2 class="font-bold text-gray-700 mb-2">当前状态</h2>  
 <p id="recording-status" class="text-gray-600">空闲</p>  
 <p id="playback-status" class="text-gray-600">无任务运行</p>  
 </div>  
  
 <a href="options.html" target="\_blank" class="w-full text-center py-2 px-4 bg-blue-500 text-white rounded-lg shadow-md hover:bg-blue-600 transition">  
 打开管理面板  
 </a>  
 </div>  
 <script src="popup.js"></script>  
</body>  
</html>

### 5.2. popup/popup.js

document.addEventListener('DOMContentLoaded', () => {  
 const toggleRecordBtn = document.getElementById('toggle-record-btn');  
 const recordingStatusEl = document.getElementById('recording-status');  
 const playbackStatusEl = document.getElementById('playback-status');  
  
 let isRecording = false;  
  
 // 更新UI的函数  
 function updateUI(state) {  
 isRecording = state.recordingState.isRecording;  
 if (isRecording) {  
 toggleRecordBtn.textContent = '停止录制';  
 toggleRecordBtn.classList.add('recording');  
 recordingStatusEl.textContent = `正在录制 Tab: ${state.recordingState.targetTabId}`;  
 } else {  
 toggleRecordBtn.textContent = '开始录制';  
 toggleRecordBtn.classList.remove('recording');  
 recordingStatusEl.textContent = '空闲';  
 }  
  
 const runningTasks = state.playbackQueue.filter(t => t.status === 'running').length;  
 playbackStatusEl.textContent = `有 ${runningTasks} 个任务正在运行`;  
 }  
  
 // 初始化时获取当前状态  
 chrome.runtime.sendMessage({  
 type: 'GET\_STATE'  
 }, updateUI);  
  
 // 监听来自 service-worker 的状态更新  
 chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {  
 if (message.type === 'STATE\_UPDATED') {  
 updateUI(message.payload);  
 }  
 });  
  
 // 按钮点击事件  
 toggleRecordBtn.addEventListener('click', async () => {  
 if (isRecording) {  
 chrome.runtime.sendMessage({  
 type: 'STOP\_RECORDING'  
 });  
 } else {  
 const [tab] = await chrome.tabs.query({  
 active: true,  
 currentWindow: true  
 });  
 chrome.runtime.sendMessage({  
 type: 'START\_RECORDING',  
 payload: {  
 tabId: tab.id,  
 url: tab.url  
 }  
 });  
 }  
 // 短暂延迟后关闭popup，避免用户看到闪烁  
 setTimeout(() => window.close(), 200);  
 });  
});

## 6. UI - 选项页面 (Options Page)

### 6.1. options/options.html

<!DOCTYPE html>  
<html>  
<head>  
 <meta charset="UTF-8">  
 <title>管理面板 - 内网助手</title>  
 <script src="[https://cdn.tailwindcss.com](https://cdn.tailwindcss.com)"></script>  
</head>  
<body class="bg-gray-50">  
 <div class="container mx-auto p-8">  
 <h1 class="text-4xl font-bold text-gray-800 mb-8">管理面板</h1>  
  
 <!-- Tab Navigation -->  
 <div class="mb-8 border-b border-gray-200">  
 <nav class="-mb-px flex space-x-8" aria-label="Tabs">  
 <a href="#scripts" class="tab active-tab">脚本管理</a>  
 <a href="#schedules" class="tab">定时任务</a>  
 <a href="#logs" class="tab">任务日志</a>  
 <a href="#settings" class="tab">全局设置</a>  
 </nav>  
 </div>  
  
 <!-- Tab Content -->  
 <div id="tab-content">  
 <!-- 脚本管理 (默认显示) -->  
 <div id="scripts" class="tab-pane">  
 <div class="flex justify-between items-center mb-6">  
 <h2 class="text-2xl font-semibold text-gray-700">我的脚本</h2>  
 <div>  
 <button id="import-btn" class="btn btn-secondary mr-2">导入脚本</button>  
 <button id="new-script-btn" class="btn btn-primary">创建新脚本</button>  
 </div>  
 </div>  
 <div id="script-list" class="bg-white p-4 rounded-lg shadow">  
 <!-- 脚本列表将动态填充在这里 -->  
 <p class="text-gray-500">正在加载脚本...</p>  
 </div>  
 </div>  
  
 <!-- 其他 Tab Pane (初始隐藏) -->  
 <div id="schedules" class="tab-pane hidden">  
 <h2 class="text-2xl font-semibold text-gray-700">定时任务</h2>  
 <!-- 定时任务内容 -->  
 </div>  
 <div id="logs" class="tab-pane hidden">  
 <h2 class="text-2xl font-semibold text-gray-700">任务日志</h2>  
 <!-- 日志内容 -->  
 </div>  
 <div id="settings" class="tab-pane hidden">  
 <h2 class="text-2xl font-semibold text-gray-700">全局设置</h2>  
 <div class="bg-white p-6 rounded-lg shadow-md mt-6">  
 <h3 class="text-lg font-bold text-yellow-600 mb-4">重要：浏览器下载设置</h3>  
 <p class="text-gray-700 mb-4">  
 为了使“自动下载并重命名”功能生效，您必须手动配置Chrome浏览器。扩展程序无法自动修改这些安全设置。  
 </p>  
 <ol class="list-decimal list-inside space-y-2 text-gray-600">  
 <li>关闭 <strong class="text-gray-800">“下载前询问每个文件的保存位置”</strong> 选项。</li>  
 <li>设置一个固定的 <strong class="text-gray-800">“下载位置”</strong> 作为默认文件夹。</li>  
 </ol>  
 <a href="chrome://settings/downloads" target="\_blank" class="mt-4 inline-block text-blue-500 hover:underline">  
 点此立即前往设置页面 →  
 </a>  
 </div>  
 <!-- 其他设置 -->  
 </div>  
 </div>  
 </div>  
  
 <style>  
 .tab {  
 @apply whitespace-nowrap py-4 px-1 border-b-2 font-medium text-sm text-gray-500 hover:text-gray-700 hover:border-gray-300;  
 }  
 .active-tab {  
 @apply border-indigo-500 text-indigo-600;  
 }  
 .btn {  
 @apply py-2 px-4 rounded-lg font-semibold shadow-sm transition-transform transform hover:scale-105;  
 }  
 .btn-primary { @apply bg-indigo-600 text-white hover:bg-indigo-700; }  
 .btn-secondary { @apply bg-white text-gray-700 border border-gray-300 hover:bg-gray-50; }  
 </style>  
 <script src="options.js"></script>  
</body>  
</html>

### 6.2. options/options.js

document.addEventListener('DOMContentLoaded', () => {  
 const tabs = document.querySelectorAll('.tab');  
 const panes = document.querySelectorAll('.tab-pane');  
  
 // Tab 切换逻辑  
 tabs.forEach(tab => {  
 tab.addEventListener('click', (e) => {  
 e.preventDefault();  
 tabs.forEach(t => t.classList.remove('active-tab'));  
 panes.forEach(p => p.classList.add('hidden'));  
  
 tab.classList.add('active-tab');  
 const targetPaneId = tab.getAttribute('href').substring(1);  
 document.getElementById(targetPaneId).classList.remove('hidden');  
 });  
 });  
  
 // --- 脚本管理 ---  
 const scriptListEl = document.getElementById('script-list');  
  
 async function loadScripts() {  
 const {  
 scripts  
 } = await chrome.storage.local.get('scripts');  
 if (!scripts || scripts.length === 0) {  
 scriptListEl.innerHTML = '<p class="text-gray-500">您还没有任何脚本。点击右上角“创建新脚本”开始吧！</p>';  
 return;  
 }  
  
 // 清空列表  
 scriptListEl.innerHTML = '';  
 // 渲染列表  
 scripts.forEach(script => {  
 const scriptEl = document.createElement('div');  
 scriptEl.className = 'flex justify-between items-center p-3 border-b last:border-b-0';  
 scriptEl.innerHTML = `  
 <div>  
 <p class="font-bold text-gray-800">${script.name}</p>  
 <p class="text-sm text-gray-500">${script.description || '无描述'}</p>  
 </div>  
 <div class="flex space-x-2">  
 <button data-script-id="${script.id}" class="run-btn btn btn-secondary">运行</button>  
 <button data-script-id="${script.id}" class="edit-btn btn btn-secondary">编辑</button>  
 <button data-script-id="${script.id}" class="delete-btn btn btn-secondary">删除</button>  
 </div>  
 `;  
 scriptListEl.appendChild(scriptEl);  
 });  
 }  
  
 // 初始加载  
 loadScripts();  
  
 // TODO: 实现创建、编辑、删除、运行脚本的事件监听和逻辑  
 // 示例：删除  
 scriptListEl.addEventListener('click', async (e) => {  
 if (e.target.classList.contains('delete-btn')) {  
 const scriptId = e.target.dataset.scriptId;  
 if (confirm('确定要删除这个脚本吗？')) {  
 let {  
 scripts  
 } = await chrome.storage.local.get('scripts');  
 scripts = scripts.filter(s => s.id !== scriptId);  
 await chrome.storage.local.set({  
 scripts  
 });  
 loadScripts(); // 重新加载列表  
 }  
 }  
 });  
  
});

## 7. Content Scripts - 页面交互的执行者

### 7.1. content/recorder.js

// content/recorder.js - 注入页面用于录制操作  
  
// 防止重复注入  
if (!window.recorderInjected) {  
 window.recorderInjected = true;  
  
 console.log("录制器已注入。");  
  
 // 在页面顶部显示一个录制状态栏  
 const recorderBar = document.createElement('div');  
 recorderBar.id = 'my-extension-recorder-bar';  
 recorderBar.style.position = 'fixed';  
 recorderBar.style.top = '0';  
 recorderBar.style.left = '0';  
 recorderBar.style.width = '100%';  
 recorderBar.style.backgroundColor = '#ef4444';  
 recorderBar.style.color = 'white';  
 recorderBar.style.textAlign = 'center';  
 recorderBar.style.padding = '10px';  
 recorderBar.style.zIndex = '9999999';  
 recorderBar.style.fontFamily = 'sans-serif';  
 recorderBar.textContent = '🔴 正在录制操作... 点击扩展图标停止录制。';  
 document.body.prepend(recorderBar);  
  
  
 const eventHandler = (e) => {  
 // 忽略对录制栏本身的操作  
 if (e.target.id === 'my-extension-recorder-bar') {  
 return;  
 }  
  
 let action = null;  
 const target = e.target;  
  
 switch (e.type) {  
 case 'click':  
 if (target.matches('input[type="file"]') || (target.closest('label')?.htmlFor && document.getElementById(target.closest('label').htmlFor)?.type === 'file')) {  
 action = {  
 type: 'uploadClick'  
 };  
 } else {  
 action = {  
 type: 'click'  
 };  
 }  
 break;  
 case 'change':  
 if (target.tagName === 'SELECT') {  
 action = {  
 type: 'select',  
 value: target.value  
 };  
 } else {  
 action = {  
 type: 'type',  
 value: target.value  
 };  
 }  
 break;  
 }  
  
 if (action) {  
 action.selector = generateSelector(target);  
 console.log("录制到操作:", action);  
 chrome.runtime.sendMessage({  
 type: 'RECORD\_ACTION',  
 payload: action  
 });  
 }  
 };  
  
 // 监听关键事件  
 ['click', 'change'].forEach(eventName => {  
 document.addEventListener(eventName, eventHandler, {  
 capture: true  
 });  
 });  
  
 // TODO: 实现滚动事件的节流(debounce)处理  
}

### 7.2. content/player.js

// content/player.js - 注入页面用于回放操作  
  
// 防止重复注入  
if (!window.playerInjected) {  
 window.playerInjected = true;  
 console.log("回放器已注入。");  
  
 chrome.runtime.onMessage.addListener((message, sender, sendResponse) => {  
 if (message.type === 'EXECUTE\_STEP') {  
 const {  
 step,  
 taskId  
 } = message.payload;  
 executeStep(step)  
 .then(result => {  
 chrome.runtime.sendMessage({  
 type: 'STEP\_COMPLETED',  
 payload: {  
 taskId,  
 result  
 }  
 });  
 });  
 }  
 });  
  
 async function executeStep(step) {  
 try {  
 let element;  
 if (step.selector) {  
 element = await waitForElement(step.selector, step.timeout || 30000);  
 }  
  
 // 执行前回放高亮  
 if (element) {  
 element.scrollIntoView({  
 behavior: 'smooth',  
 block: 'center'  
 });  
 element.style.outline = '3px solid #8b5cf6';  
 await new Promise(r => setTimeout(r, 500)); // 等待用户看到高亮  
 }  
  
  
 switch (step.type) {  
 case 'click':  
 case 'uploadClick': // uploadClick 和 click 的初始动作一样  
 element.click();  
 break;  
 case 'type':  
 element.value = step.value;  
 element.dispatchEvent(new Event('input', {  
 bubbles: true  
 }));  
 element.dispatchEvent(new Event('change', {  
 bubbles: true  
 }));  
 break;  
 case 'select':  
 element.value = step.value;  
 element.dispatchEvent(new Event('change', {  
 bubbles: true  
 }));  
 break;  
 case 'scroll':  
 // TODO: 实现滚动逻辑  
 break;  
 case 'wait':  
 if (step.waitType === 'time') {  
 await new Promise(r => setTimeout(r, step.value));  
 } else if (step.waitType === 'elementVisible') {  
 await waitForElement(step.selector, step.timeout);  
 }  
 break;  
 }  
  
 if (element) {  
 await new Promise(r => setTimeout(r, 200)); // 动作后短暂延迟  
 element.style.outline = ''; // 移除高亮  
 }  
  
 return {  
 success: true  
 };  
 } catch (error) {  
 console.error("回放步骤失败:", step, error);  
 return {  
 success: false,  
 error: error.message  
 };  
 }  
 }  
  
 function waitForElement(selector, timeout) {  
 return new Promise((resolve, reject) => {  
 const element = document.querySelector(selector);  
 if (element) {  
 resolve(element);  
 return;  
 }  
  
 const observer = new MutationObserver(() => {  
 const el = document.querySelector(selector);  
 if (el) {  
 observer.disconnect();  
 resolve(el);  
 }  
 });  
  
 const timer = setTimeout(() => {  
 observer.disconnect();  
 reject(new Error(`元素未找到: ${selector}`));  
 }, timeout);  
  
 observer.observe(document.body, {  
 childList: true,  
 subtree: true  
 });  
 });  
 }  
}

## 8. 工具函数 (utils/)

### 8.1. utils/storage.js

// 封装 chrome.storage.local API  
export const Storage = {  
 async get(keys = null) {  
 return await chrome.storage.local.get(keys);  
 },  
 async set(data) {  
 return await chrome.storage.local.set(data);  
 },  
 async remove(keys) {  
 return await chrome.storage.local.remove(keys);  
 },  
 async clear() {  
 return await chrome.storage.local.clear();  
 },  
};

### 8.2. utils/uuid.js

export function generateUUID() {  
 return crypto.randomUUID();  
}

### 8.3. utils/url\_parser.js

export function resolveUrl(template, variables = {}) {  
 let resolved = template;  
 // 内置宏  
 resolved = resolved.replace(/{DATE\_TODAY}/g, new Date().toISOString().slice(0, 10));  
 resolved = resolved.replace(/{TIME\_HHMM}/g, new Date().toTimeString().slice(0, 5).replace(':', ''));  
  
 // 动态变量  
 for (const [key, value] of Object.entries(variables)) {  
 const placeholder = new RegExp(`{${key}}`, 'g');  
 resolved = resolved.replace(placeholder, value);  
 }  
 return resolved;  
}

### 8.4. utils/selector.js

// 此文件需要被注入到 content script 中，因此不能使用 ES 模块语法  
function generateSelector(element) {  
 if (!element) return '';  
 // 优先级 1: data-testid, data-cy 等专用测试属性  
 for (const attr of ['data-testid', 'data-cy', 'data-test-id']) {  
 if (element.hasAttribute(attr)) {  
 return `[${attr}="${element.getAttribute(attr)}"]`;  
 }  
 }  
  
 // 优先级 2: 唯一的 ID  
 if (element.id && document.querySelectorAll(`#${CSS.escape(element.id)}`).length === 1) {  
 return `#${CSS.escape(element.id)}`;  
 }  
  
 // 优先级 3: 唯一的 Name 属性 (常用于表单)  
 if (element.name && element.tagName !== 'FORM' && document.querySelectorAll(`[name="${CSS.escape(element.name)}"]`).length === 1) {  
 return `[name="${CSS.escape(element.name)}"]`;  
 }  
  
 // 备用方案: 构建CSS Path  
 let path = [];  
 let current = element;  
 while (current && current.nodeType === Node.ELEMENT\_NODE && current.tagName.toLowerCase() !== 'html') {  
 let selector = current.tagName.toLowerCase();  
 if (current.id) {  
 selector += `#${CSS.escape(current.id)}`;  
 path.unshift(selector);  
 break;  
 }  
   
 let siblingIndex = 1;  
 let sibling = current.previousElementSibling;  
 while (sibling) {  
 if (sibling.tagName === current.tagName) {  
 siblingIndex++;  
 }  
 sibling = sibling.previousElementSibling;  
 }  
   
 if (current.parentElement.querySelectorAll(`${selector}`).length > 1) {  
 selector += `:nth-of-type(${siblingIndex})`;  
 }  
  
 path.unshift(selector);  
 current = current.parentElement;  
 }  
 return path.join(' > ');  
}