官方提供的插件脚手架的入口为 `main.ts` 中继承了 `Plugin` 的子类

撒旦
注：obsidian 是半开源的（仅开放了插件相关接口）

参考

集成 vue3

`npm i -D vue esbuild-plugin-vue3`

`esbuild.config.mjs`：

```
// (1) vue 配置
import esbuildPluginVue3 from "esbuild-plugin-vue3"
 ...
const context = await esbuild.context({
         ...
        plugins: [esbuildPluginVue3()]
})
 ...

// (2) css 相关配置
import fs from 'fs'
const prod = (process.argv[2] === "production")
if (prod) {
        await context.rebuild();
        fs.rename("main.css", "styles.css", (err) => {
                if (err) {
                        throw err;
                }
        })
        process.exit(0);
} else {
        await context.watch();
        fs.watchFile("main.css", () => {
                fs.access("main.css", fs.constants.F_OK, (err) => {
                        if (!err) {
                                fs.rename("main.css", "styles.css", (err) =>
{
                                        if (err) {
                                                throw err
                                        }
                                })
                        }
                })
        })
}
```

挂载 vue 组件的例子：

```
import {createApp} from 'vue'
mount(el: HTMLElement, sfc: Component){
        createApp(sfc).mount(el)
}
```

核心

app

`obsidian` => `App`

`App` 类：

```
{
        // 工作区
        workspace: Workspace,
        // 仓库
        vault: Vault,
        // 文件管理
        fileManager: FileManager,
        // 元数据
        metadataCache: MetadataCache,

        keymap: Keymap,
        scope: Scope,
        lastEvent: UserEvent | null
}
```

元素

`obsidian` => `Node`, `Element`, `HTMLElement`

`Node` 接口：

```
{
        doc: Document;
        win: Window;

        detach(): void;
        empty(): void;
        insertAfter<T extends Node>(node: T, child: Node | null): T;
        indexOf(other: Node): num;
        setChildrenInPlace(children: Node[]): void;
        appendText(val: str): void;
        instanceOf<T>(type: {
                new (): T;
        }): this is T;
        constructorWin: Window;
}

{
        createEl<K extends keyof HTMLElementTagNameMap>(tag: K, o?:
DomElementInfo | str, cb?: (el: HTMLElementTagNameMap[K]) => void):
HTMLElementTagNameMap[K];
        createDiv(o?: DomElementInfo | str, cb?: (el: HTMLDivElement) =>
void): HTMLDivElement;
        createSpan(o?: DomElementInfo | str, cb?: (el: HTMLSpanElement) =>
void): HTMLSpanElement;
        createSvg<K extends keyof SVGElementTagNameMap>(tag: K, o?:
SvgElementInfo | str, cb?: (el: SVGElementTagNameMap[K]) => void):
SVGElementTagNameMap[K];
}
```

`Element < Node` 接口:

```
{
        getText(): str;
        setText(val: str | DocumentFragment): void;
        addClass( ... classes: str[]): void;
        addClasses(classes: str[]): void;
        removeClass( ... classes: str[]): void;
        removeClasses(classes: str[]): void;
        toggleClass(classes: str | str[], value: bool): void;
        hasClass(cls: str): bool;
        setAttr(qualifiedName: str, value: str | num | bool | null): void;
        setAttrs(obj: {
                [key: str]: str | num | bool | null;
        }): void;
        getAttr(qualifiedName: str): str | null;
        matchParent(selector: str, lastParent?: Element): Element | null;
        getCssPropertyValue(property: str, pseudoElement?: str): str;
        isActiveElement(): bool;
}
```

`HTMLElement < Element` 接口:

```
{
        show(): void;
        hide(): void;
        toggle(show: bool): void;
        toggleVisibility(visible: bool): void;
        isShown(): bool;
        setCssStyles(styles: Partial<CSSStyleDeclaration>): void;
        setCssProps(props: Record<str, str>): void;
        readonly innerWidth: num;
        readonly innerHeight: num;
}
```

插件

`obsidian` => `Plugin`, `Component`

`Component` 类:

```
{
        abstract onload: ()⇒void,
        abstract onunload: ()⇒void,

        load(): void,
        unload(): void,

        // 注册 dom 事件
        registerDomEvent(el: Window | Document | HTMLElement, type: enum,
cb: (this: HTMLElement, e: Event) ⇒ any, opts?: bool |
AddEventListenerOptions): void,
        // 注册可由 obsidian 管理的事件，如： workspace 的 file-open 事件
        registerEvent(eventRef: EventRef): void,
        // 跟踪 window.setInterval() 注册到的周期任务，在插件不可用时销毁该任务
        registerInterval(id: num): num,
        // 注册一个 unloading 的回调
        register(cb: () ⇒ any): void,

        addChild<T extends Component>(component: T): T,
        removeChild<T extends Component>(component: T): T,
}
```

`Plugin < Component` 类:

```
{
        app: App,
        manifest: PluginManifest,

        // 在 左边栏 添加一个 ribbon icon
        addRibbonIcon(iconId: IconName, title: str, cb: (e: MouseEvent) ⇒
any): HTMLElement,
        // 在 底栏 添加一个 status bar 条目（手机端不可用）
        addStatusBarItem(): HTMLElement,
        // 注册一个 命令
        addCommand(cmd: Command): Command,
        // 注册一个 settings tab
        addSettingTab(tab: PluginSettingTab): void,
        // 注册一个 view
        registerView(type: str, viewCreator: ViewCreator): void,

        // 加载 data.json 文件中的数据
        loadData(): Promise<any>,
        // 将数据保存到 data.json
        saveData(data: any): Promise<void>,

        // 注册 markdown 后处理器（在 阅读模式 下可以对源码进行再次渲染）
        registerMarkdownPostProcessor(postProcessor: MarkdownPostProcessor,
sortOrder?: num): MarkdownPostProcessor,
        // 注册 markdown 代码块处理器（在 阅读/实时模式 下可以对代码块进行再次渲染）
        registerMarkdownCodeBlockProcessor(language: str, handler: (source:
str, el: HTMLElement, ctx: MarkdownPostProcessorContext) => Promise<any> |
void, sortOrder?: num): MarkdownPostProcessor,
        //
        registerEditorSuggest(editorSuggest: EditorSuggest<any>): void,
        registerEditorExtension(extension: Extension): void,

        registerHoverLinkSource(id: str, info: HoverLinkSource): void,
        registerExtensions(extensions: str[], viewType: str): void,
        registerObsidianProtocolHandler(action: str, handler:
ObsidianProtocolHandler): void,

        onExternalSettingsChange?(): any
}
```

例子：

```
// dom 事件
this.registerDomEvent(document, 'click',(e)=>console.log(e))
// obsidian 事件
this.registerEvent(this.app.workspace.on('file-open',(file)=>{
        console.log(file)
}))
// 周期性任务
this.registerInterval(window.setInterval(()=>{
        console.log('interval task')
}, 1000))
```

ui

左边栏/底栏

例子：

```
export default class MyPlugin extends Plugin{
        async onload(){
                // 左边栏
                let rb = this.addRibbonIcon('bar-chart-horizontal-big',
'example ribbon icon', (e)=> {
                        new Notice('example ribbon icon')
                })

                // 底栏
                let sb = this.addStatusBarItem()
                sb.setText('example status bar item')
                // sb.createEl('button', '', (el)=>{
                //      el.innerHTML = 'modal'
                //      el.onclick=()=>this.modal.open()
                // })
        }
}
```

命令

`obsidian` => `Command`, `Hotkey`, `Modifier`

`Command` 接口:

```
{
        // 命令的全局唯一 id
        id: str,
        // 人类友好名称
        name: str,
        hotkeys?: Hotkey[],

        // 简单回调
        callback?: ()=>any,
        // 在触发回调前，执行一次是否调用该回调的检查
        checkCallback?: (checking: bool) => bool | void,
        // 仅在编辑器中可触发回调
        editorCallback?: (edt: Editor, ctx: MarkdownView | MarkdownFileInfo)
=> any,
        editorCheckCallback?: (checking: bool, edt: Editor, ctx:
MarkdownView | MarkdownFileInfo) => bool | void,

        icon?: IconName,
        mobileOnly?: bool,
        repeatable?: bool,
}
```

`Hotkey` 接口:

```
{
        modifiers: Modifier[],
        key: str
}
```

`Modifier` 类型 => `Modifier: 'Mod' | 'Ctrl' | 'Meta' | 'Shift' | 'Alt'`

例子:

```
// (1) 简单命令
let cmd = {
        id: 'example-cmd',
        name: 'example cmd',
        callback: ()=>{
                // do something ...
        }
}

// (2) 回调前预检查
let cmd = {
        id: 'example-cmd',
        name: 'example cmd',
        checkCallback: (checking: bool) => {
                const view =
this.app.workspace.getActiveViewOfType(MarkdownView)
                if(view){
                        if(!checking){
                                // do something ...
                        }
                        return true
                }
                return false
        }
}

// (3) 可修改编辑器的命令
let cmd = {
        id: 'example-cmd',
        name: 'example cmd',
        editorCallback: (editor: Editor, view: MarkdownView) => {
                // do something ...
        }
}
```

设置

`obsidian` => `SettingTab`, `PluginSettingTab`, `Setting`

`SettingTab` 抽象类:

```
{
        // 显示 配置项 tab 时的回调
        abstract display(): any,
        hide(): any,

        app: App,
        // 用于操作 配置项 的 ui
        containerEl: HTMLElement,
}
```

`PluginSettingTab < SettingTab` 抽象类:

```
{
        constructor(app: App, plugin: Plugin)
}
```

`Setting` 类:

```
{
        constructor(containerEl: HTMLElement),

        settingEl: HTMLElement,
        infoEl: HTMLElement,
        nameEl: HTMLElement,
        descEl: HTMLElement,
        controlEl: HTMLElement,
        components: BaseComponent[],

        setName(name: str | DocumentFragment): this,
        setDesc(desc: str | DocumentFragment): this,
        setClass(cls: str): this,
        setTooltip(tooltip: str, options?: TooltipOptions): this,
        setHeading(): this,
        setDisabled(disabled: bool): this,

        addButton(cb: (component: ButtonComponent) => any): this,
        addExtraButton(cb: (component: ExtraButtonComponent) => any): this,
        addToggle(cb: (component: ToggleComponent) => any): this,
        addText(cb: (component: TextComponent) => any): this;
    addSearch(cb: (component: SearchComponent) => any): this,
    addTextArea(cb: (component: TextAreaComponent) => any): this,
    addMomentFormat(cb: (component: MomentFormatComponent) => any): this,
    addDropdown(cb: (component: DropdownComponent) => any): this,
    addColorPicker(cb: (component: ColorComponent) => any): this,
    addProgressBar(cb: (component: ProgressBarComponent) => any): this,
    addSlider(cb: (component: SliderComponent) => any): this,

    then(cb: (setting: this) => any): this,
    clear(): this
}
```

例子:

```
class MyPluginSettingTab extends PluginSettingTab {
        plugin: MyPlugin
        constructor(app: App, plugin: MyPlugin) {
                super(app, plugin)
                this.plugin = plugin
        }
        async display() {
                await this.plugin.load_settings()
        }
}

interface MyPluginSettings {
        example_setting: str;
}
const DEFAULT_SETTINGS: MyPluginSettings = {
        example_setting: 'default'
}

export default class MyPlugin extends Plugin{
        settingTab: MyPluginSettingTab;
        settings: MyPluginSettings;
        async load_settings(){
                // 清理过期的配置项 ui
                this.settingTab.containerEl.empty()

                // 加载配置项，并填充配置项 ui 的数据
                this.settings = Object.assign({}, DEFAULT_SETTINGS, await
this.loadData())
                this.addSettingTab(this.settingTab)
                new Setting(this.settingTab.containerEl)
                        .setName('example setting')
                        .setDesc('example description')
                        .addText((text) => {
                                text.setPlaceholder('example placeholder')

.setValue(this.settings.example_setting)
                                        .onChange(async (val)=>{

this.settings.example_setting = val
                                                await
this.saveData(this.settings)
```

```
                                        })
                        })
        }

        async onload(){
                // 初始化配置项 ui
                this.settingTab = new MyPluginSettingTab(this.app, this)
                await this.load_settings()
        }
}
```

注：这个方式只会在加载插件时读取 `data.json`

上下文菜单

`obsidian` => `Menu`, `MenuItem`

`Menu < Component, CloseableComponent` 类：

```
{
        constructor(),

        setNoIcon(): this,
        setUseNativeMenu(useNativeMenu: bool): this;

        // 添加 menu item
    addItem(cb: (item: MenuItem) ⇒ any): this,
    // 添加分隔符
    addSeparator(): this,

    // 在鼠标事件处显示上下文菜单
    showAtMouseEvent(evt: MouseEvent): this,
    showAtPosition(pos: MenuPositionDef, doc?: Document): this,

    hide(): this,
    close(): void,
    onHide(cb: () ⇒ any): void
}
```

`MenuItem` 类：

```
{
    private constructor();

    setTitle(title: str | DocumentFragment): this,
    setIcon(icon: IconName | null): this,
    setChecked(checked: bool | null): this,
    setDisabled(disabled: bool): this,
    setIsLabel(isLabel: bool): this,
    onClick(cb: (evt: MouseEvent | KeyboardEvent) => any): this,
    setSection(section: str): this
}
```

例子:

```
export default class MyPlugin extends Plugin{
    ribbon_menu: Menu;
    init_menu(){
        this.ribbon_menu = new Menu()
        this.ribbon_menu.addItem((item)=>{
            item.setTitle('example menu item')
                .onClick(()=>new Notice('example menu item'))
        })

        this.addRibbonIcon('dice', 'example ribbon icon', (e)=>{
            this.ribbon_menu.showAtMouseEvent(e)
        })
    }

    async onload(){
        this.init_menu()
    }
}
```

模态

`obsidian` => `Modal`

`Modal` 类:

```
{
  constructor(app: App);

  onOpen(): void,
  onClose(): void,
  open(): void,
  close(): void,

  app: App,
  scope: Scope,

  // 模态的 html 元素
  containerEl: HTMLElement,
  modalEl: HTMLElement,
  titleEl: HTMLElement,
    contentEl: HTMLElement,

    shouldRestoreSelection: bool,

    // 标题
    setTitle(title: str): this,
    // 内容
    setContent(content: str | DocumentFragment): this
}
```

例子：

```
export default class MyPlugin extends Plugin{
        modal: MyModal;
        init_modal(){
                this.modal = new MyModal(this.app)
                this.modal
                        .setTitle('example modal')
                        .setContent('example modal content')

                this.addRibbonIcon('dice', 'example ribbon icon', (e)=>{
                        this.modal.open()
                })
        }

        async onload(){
                this.init_modal()
        }
}
```

通知

`obsidian` => `Notice`

`Notice` 类：

```
{
    constructor(message: str | DocumentFragment, duration?: num);
    noticeEl: HTMLElement;
    setMessage(message: str | DocumentFragment): this;
    hide(): void;
}
```

例子：

```
class MyNotice extends Notice{
        constructor(cb: (el: HTMLElement)=>void, duration?: number) {
                super('', duration)
                cb(this.noticeEl)
        }
}

export default class MyPlugin extends Plugin{
        async onload(){
                // 与 new Notice('example notice') 等效
                new MyNotice((el)⇒el.setText('example notice'))
        }
}
```

图标

`obsidian` => `addIcon()`

```
addIcon(iconId: str, svgContent: str): void
```

工作区

`obsidian` => `Workspace`, `WorkspaceItem`, `WorkspaceParent`, `WorkspaceLeaf`, `WorkspaceSplit`, `WorkspaceTabs`

核心:

- `Workspace`, `WorkspaceItem` < `Events`
- `WorkspaceParent`, `WorkspaceLeaf` < `WorkspaceItem`
  `parent` 家族:
- `WorkspaceSplit`, `WorkspaceTabs`, `WorkspaceFloating`, `WorkspaceMobileDrawer` < `WorkspaceParent`
- `WorkspaceContainer`, `WorkspaceSidedock` < `WorkspaceSplit`
- `WorkspaceRoot`, `WorkspaceWindow` < `WorkspaceContainer`
  注: `WorkspaceSplit`、`WorkspaceTabs`、`WorkspaceFloating` 是空的

`WorkspaceRibbon`

注意区分:

- `leaf` / `parent`
- `tabs` / `split`
  重点 => `Workspace`, `WorkspaceLeaf`

`Events` 类:

```
{
    on(name: str, cb: ( ... data: any) ⇒ any, ctx?: any): EventRef;
    off(name: str, cb: ( ... data: any) ⇒ any): void;
    offref(ref: EventRef): void;
    trigger(name: str,  ... data: any[]): void;
    tryTrigger(evt: EventRef, args: any[]): void;
}
```

`Workspace < Events` 类：

```
{
        // rootSplit 可进行 vertical/horizontal split
    rootSplit: WorkspaceRoot,
    // leftSplit 和 rightSplit 只能进行 horizontal split
        leftSplit: WorkspaceSidedock | WorkspaceMobileDrawer;
    rightSplit: WorkspaceSidedock | WorkspaceMobileDrawer,

    containerEl: HTMLElement,

    leftRibbon: WorkspaceRibbon,
    rightRibbon: WorkspaceRibbon,

    requestSaveLayout: Debouncer<[], Promise<void>>,
    activeEditor: MarkdownFileInfo | null,

        // ==== (1) create leaf

    // 对 rootSplit 中最近聚焦的 leaf 所在的容器进行分裂（向右 或 向下 分裂）
    getLeaf(newLeaf?: 'split', direction?: 'vertical' | 'horizontal'):
WorkspaceLeaf,
    // 对 rootSplit 中最近聚焦的 leaf 进行各种新建 leaf 的操作
    getLeaf(newLeaf?: 'tab' | 'split' | 'window' | bool): WorkspaceLeaf,
    getLeftLeaf(split: bool): WorkspaceLeaf | null,
    getRightLeaf(split: bool): WorkspaceLeaf | null,
    // 创建弹出式 leaf
    openPopoutLeaf(data?: WorkspaceWindowInitData): WorkspaceLeaf,
    // 通过分裂 leaf 来创建新 leaf
    createLeafBySplit(leaf: WorkspaceLeaf, direction?: SplitDirection,
before?: bool): WorkspaceLeaf,
    // 在父 split 中分裂一个新 leaf
    createLeafInParent(parent: WorkspaceSplit, index: number):
WorkspaceLeaf,

    // ==== (2) retrieve leaf
    getLeafById(id: str): WorkspaceLeaf | null,
    getMostRecentLeaf(root?: WorkspaceParent): WorkspaceLeaf | null,
    getLeavesOfType(viewType: str): WorkspaceLeaf[],
    getGroupLeaves(group: str): WorkspaceLeaf[],

    activeLeaf: WorkspaceLeaf | null,

    // ==== (3) update leaf
    splitActiveLeaf(direction?: SplitDirection): WorkspaceLeaf,
    // 将普通 leaf 转换为弹出式 leaf
    moveLeafToPopout(leaf: WorkspaceLeaf, data?: WorkspaceWindowInitData):
WorkspaceWindow,

        // 显示 leaf
    revealLeaf(leaf: WorkspaceLeaf): void,
        // 根据 linktext 打开显示文件的 leaf
    openLinkText(linktext: str, sourcePath: str, newLeaf?: PaneType | bool,
openViewState?: OpenViewState): Promise<void>,

    // 迭代打开的 leaf
    iterateAllLeaves(cb: (leaf: WorkspaceLeaf) => any): void,
    // 迭代打开的 root leaf
    iterateRootLeaves(cb: (leaf: WorkspaceLeaf) => any): void,
    detachLeavesOfType(viewType: str): void,

    // 设置 active leaf
    setActiveLeaf(leaf: WorkspaceLeaf, params?: { focus?: bool }): void,
    setActiveLeaf(leaf: WorkspaceLeaf, pushHistory: bool, focus: bool):
void,

        changeLayout(workspace: any): Promise<void>;
    getLayout(): any,
    duplicateLeaf(leaf: WorkspaceLeaf, direction?: SplitDirection):
Promise<WorkspaceLeaf>,
    duplicateLeaf(leaf: WorkspaceLeaf, leafType: PaneType | bool,
direction?: SplitDirection): Promise<WorkspaceLeaf>,
    getUnpinnedLeaf(): WorkspaceLeaf,
    getActiveViewOfType<T extends View>(type: Constructor<T>): T | null,
    getActiveFile(): TFile | null,
    getLastOpenFiles(): str[],
    updateOptions(): void,

    // 事件相关回调、属性
    layoutReady: bool,
        onLayoutReady(cb: () => any): void,
    on(name: 'quick-preview', cb: (file: TFile, data: str) => any, ctx?:
any): EventRef,
    on(name: 'resize', cb: () => any, ctx?: any): EventRef,
    on(name: 'active-leaf-change', cb: (leaf: WorkspaceLeaf | null) => any,
ctx?: any): EventRef,
    on(name: 'file-open', cb: (file: TFile | null) => any, ctx?: any):
EventRef,
    on(name: 'layout-change', cb: () => any, ctx?: any): EventRef,
    on(name: 'window-open', cb: (win: WorkspaceWindow, window: Window) =>
any, ctx?: any): EventRef,
    on(name: 'window-close', cb: (win: WorkspaceWindow, window: Window) =>
any, ctx?: any): EventRef,
    on(name: 'css-change', cb: () => any, ctx?: any): EventRef,
    on(name: 'file-menu', cb: (menu: Menu, file: TAbstractFile, source: str,
leaf?: WorkspaceLeaf) => any, ctx?: any): EventRef,
    on(name: 'files-menu', cb: (menu: Menu, files: TAbstractFile[], source:
str, leaf?: WorkspaceLeaf) => any, ctx?: any): EventRef,
    on(name: 'url-menu', cb: (menu: Menu, url: str) => any, ctx?: any):
EventRef,
    on(name: 'editor-menu', cb: (menu: Menu, editor: Editor, info:
MarkdownView | MarkdownFileInfo) => any, ctx?: any): EventRef,
    on(name: 'editor-change', cb: (editor: Editor, info: MarkdownView |
MarkdownFileInfo) => any, ctx?: any): EventRef,
    on(name: 'editor-paste', cb: (evt: ClipboardEvent, editor: Editor, info:
MarkdownView | MarkdownFileInfo) => any, ctx?: any): EventRef,
    on(name: 'editor-drop', cb: (evt: DragEvent, editor: Editor, info:
MarkdownView | MarkdownFileInfo) => any, ctx?: any): EventRef,
    on(name: 'quit', cb: (tasks: Tasks) => any, ctx?: any): EventRef
}
```

`WorkspaceItem < Events` 抽象类：

```
{
        getRoot(): WorkspaceItem,
        getContainer(): WorkspaceContainer
}
```

`WorkspaceLeaf < WorkspaceItem` 类：

```
{
    view: View,

    // ==== (2) retrieve
    getViewState(): ViewState,
    // 获取图标标识 id
    getIcon(): IconName,
    getDisplayText(): str,

        // ==== (3) update
    openFile(file: TFile, openState?: OpenViewState): Promise<void>,
    open(view: View): Promise<View>,
    // 更新 leaf 的状态，如：setViewState({type: 'markdown', state: {file:
'word.md'}}) => 打开 .md 文件
    setViewState(viewState: ViewState, eState?: any): Promise<void>,

    // 创建 group
    setGroup(group: str): void,
        // 添加别的 leaf 和自己组成 group
    setGroupMember(other: WorkspaceLeaf): void

    // 锁定/取消锁定
    togglePinned(): void,
    setPinned(pinned: bool): void,

    // ===== (4) delete
    detach(): void,

    getEphemeralState(): any,
    setEphemeralState(state: any): void,

    onResize(): void,
    on(name: 'pinned-change', callback: (pinned: bool) => any, ctx?: any):
EventRef,
    on(name: 'group-change', callback: (group: str) => any, ctx?: any):
EventRef,
}
```

`WorkspaceMobileDrawer < WorkspaceParent` 类：

```
{
    collapse(): void,
    expand(): void,
    toggle(): void,
    collapsed: bool,
}
```

`WorkspaceContainer < WorkspaceSplit` 抽象类：

```
{
        abstract win: Window,
        abstract doc: Document
}
```

`WorkspaceSidedock < WorkspaceSplit` 类：

```
{
    collapse(): void,
    expand(): void,
    toggle(): void,
        collapsed: bool
}
```

`WorkspaceRoot < WorkspaceContainer` 类：

```
{
    win: Window,
    doc: Document
}
```

`WorkspaceWindow < WorkspaceContainer` 类：

```
{
    win: Window,
    doc: Document
}
```

`PaneType` 类型：

```
PaneType = 'tab' | 'split' | 'window'
```

例子：

```
this.addRibbonIcon('dice', '遍历每个 leaf',async (e)=>{
        this.app.workspace.iterateAllLeaves((leaf)=>{
                console.log(leaf.getViewState())
        })
})
```

视图

`obsidian` => `View`, `ItemView`, `FileView`, `EditableFileView`, `TextFileView`
`obsidian` => `MarkdownView`, `MarkdownPreviewView`, `MarkdownEditView`

`View < Component` 抽象类：

```
{
    constructor(leaf: WorkspaceLeaf),

        app: App,
    icon: IconName,
    navigation: bool,
    leaf: WorkspaceLeaf,
    scope: Scope | null,
    // view 的容器元素
    containerEl: HTMLElement,

        // ===== 核心方法
        // 当前 view 的类型
    abstract getViewType(): str,
        // 当前 view 的 label
    abstract getDisplayText(): str,
    // open/close 的回调
    protected onOpen(): Promise<void>,
    protected onClose(): Promise<void>,

    getState(): any,
    getIcon(): IconName,

    getEphemeralState(): any,
    setEphemeralState(state: any): void,

    onResize(): void,
    onPaneMenu(menu: Menu, source: 'more-options' | 'tab-header' | str):
void
}
```

`ItemView < View` 抽象类:

```
{
        constructor(leaf: WorkspaceLeaf),

        // view 的内容元素
        contentEl: HTMLElement,
        // 在 contentEl 的标题栏中添加 action 图标
        // 注: contentEl.children[0] 即为
        addAction(icon: IconName, title: str, callback: (evt: MouseEvent) ⇒
any): HTMLElement
}
```

`FileView < ItemView` 抽象类:

```
{
    constructor(leaf: WorkspaceLeaf),

    onload(): void,
    onLoadFile(file: TFile): Promise<void>,
    onUnloadFile(file: TFile): Promise<void>,
    onRename(file: TFile): Promise<void>,

    getDisplayText(): str,

    allowNoFile: bool,
    file: TFile | null,
    navigation: bool,
    getState(): any,
    setState(state: any, result: ViewStateResult): Promise<void>,
    canAcceptExtension(extension: str): bool,
}
```

`EditableFileView < FileView`
`TextFileView < EditableFileView` 抽象类：

```
{
    constructor(leaf: WorkspaceLeaf),

        data: str,
    requestSave: () ⇒ void,
    onUnloadFile(file: TFile): Promise<void>,
    onLoadFile(file: TFile): Promise<void>,
    save(clear?: bool): Promise<void>,
    abstract getViewData(): str,
    abstract setViewData(data: str, clear: bool): void,
    abstract clear(): void,
}
```

`MarkdownView < TextFileView, MarkdownFileInfo` 类：

```
{
        editor: Editor,
    previewMode: MarkdownPreviewView,
    currentMode: MarkdownSubView,
    hoverPopover: HoverPopover | null,
    constructor(leaf: WorkspaceLeaf),
    getViewType(): str,
    getMode(): MarkdownViewModeType,
    getViewData(): str,
    clear(): void,
    setViewData(data: str, clear: bool): void,
    showSearch(replace?: bool): void,
}
```

后端

vault

`obsidian` => `Vault`

`Vault < Events` 类：

```
{
    // vault 的名字
    getName(): str,
    // vault 的配置目录，如：'.obsidian'
    configDir: str,

        // ==== (1) create
    // 创建文本文件
    create(path: str, data: str, options?: DataWriteOptions):
Promise<TFile>,
    // 创建目录
    createFolder(path: str): Promise<TFolder>,
    // 创建二进制文件
    createBinary(path: str, data: ArrayBuffer, options?: DataWriteOptions):
Promise<TFile>,

    // ===== (2) retrieve
    // 获取文件（path 为相对于 vault 的目录）
    getFileByPath(path: str): TFile | null,
    // 获取目录
    getFolderByPath(path: str): TFolder | null,
    // 获取 文件 或 目录
    getAbstractFileByPath(path: str): TAbstractFile | null,
    // 获取根目录，即 getFolderByPath('/')
    getRoot(): TFolder,
    // 获取所有 文件 和 目录
    getAllLoadedFiles(): TAbstractFile[],
    // 获取所有 .md 文件
    getMarkdownFiles(): TFile[],
    // 获取所有文件
    getFiles(): TFile[],

    // 以缓存方式读取文件
    cachedRead(file: TFile): Promise<str>,
    // 读取文件，得到文本
    read(file: TFile): Promise<str>,
    // 读取文件，得到二进制数据
    readBinary(file: TFile): Promise<ArrayBuffer>,
    // 获取文件的 URI，如：
app://e7c81fc115c6e381a95c4123e68ee034d0ea/E:/obsidian-
vault/assets/Pasted%20Image%20240521113912_615.png?1716262752639，可用于嵌
入图片等
    getResourcePath(file: TFile): str,

    // ===== (3) update
    // 修改文本文件
    modify(file: TFile, data: str, options?: DataWriteOptions):
Promise<void>,
    // 给文本文件追加文本
    append(file: TFile, data: str, options?: DataWriteOptions):
Promise<void>,
    // 以二进制方式修改文件
    modifyBinary(file: TFile, data: ArrayBuffer, options?:
DataWriteOptions): Promise<void>,

    // 复制文件（注：newPath 是一个文件，如：path/to/test.md）
    copy(file: TFile, newPath: str): Promise<TFile>,
    // 移动文件/目录
    rename(file: TAbstractFile, newPath: str): Promise<void>,

    // ===== (4) delete
    // 将文件丢入 vault 的回收站 或 系统回收站
    trash(file: TAbstractFile, system: bool): Promise<void>,
    // 彻底删除文件
    delete(file: TAbstractFile, force?: bool): Promise<void>,


    // 读取、修改、保存 纯文本文件
    process(file: TFile, fn: (data: str) => str, options?:
DataWriteOptions): Promise<str>,
    // 递归搜索目录下的所有 文件 和 目录
    static recurseChildren(root: TFolder, cb: (file: TAbstractFile) => any):
void,

    adapter: DataAdapter,

    // 文件的增删改操作的回调
    on(name: 'create', cb: (file: TAbstractFile) => any, ctx?: any):
EventRef,
    on(name: 'modify', cb: (file: TAbstractFile) => any, ctx?: any):
EventRef,
    on(name: 'delete', cb: (file: TAbstractFile) => any, ctx?: any):
EventRef,
    on(name: 'rename', cb: (file: TAbstractFile, oldPath: str) => any, ctx?:
any): EventRef
}
```

`DataAdapter` 接口：

```
// 直接操作 vault 中的 文件/目录，但推荐使用 Vault api
{
        getName(): str,
    exists(normalizedPath: str, sensitive?: bool): Promise<bool>,
    stat(normalizedPath: str): Promise<Stat | null>,
    list(normalizedPath: str): Promise<ListedFiles>,
    read(normalizedPath: str): Promise<str>,
    readBinary(normalizedPath: str): Promise<ArrayBuffer>,
    write(normalizedPath: str, data: str, options?: DataWriteOptions):
Promise<void>,
    writeBinary(normalizedPath: str, data: ArrayBuffer, options?:
DataWriteOptions): Promise<void>,
    append(normalizedPath: str, data: str, options?: DataWriteOptions):
Promise<void>,
    process(normalizedPath: str, fn: (data: str) => str, options?:
DataWriteOptions): Promise<str>,
    getResourcePath(normalizedPath: str): str,
    mkdir(normalizedPath: str): Promise<void>,
    trashSystem(normalizedPath: str): Promise<bool>,
    trashLocal(normalizedPath: str): Promise<void>,
    rmdir(normalizedPath: str, recursive: bool): Promise<void>,
    remove(normalizedPath: str): Promise<void>,
    rename(normalizedPath: str, normalizedNewPath: str): Promise<void>,
    copy(normalizedPath: str, normalizedNewPath: str): Promise<void>
}
```

文件管理

`obsidian` => `FileManager`, `TAbstractFile`, `TFolder`, `TFile`

`FileManager` 类:

```
{
    // 安全移动文件，并更新基于用户首选项的链接
    renameFile(file: TAbstractFile, newPath: str): Promise<void>,

    // 获取在 sourcePath 中指向到 file 的链接
    // 其中 subpath 为链接到 block 或 heading 的链接，如：'#example-subpath'
    // alias 为链接的别名，如：`example-alias`
    generateMarkdownLink(file: TFile, sourcePath: str, subpath?: str,
alias?: str): str,
    // 读取并解析文本的 front matter，即 .md 文本首部的 yaml 文本
    // 注：可能抛出 YAMLParseError 错误
    processFrontMatter(file: TFile, fn: (frontmatter: any) ⇒ void,
options?: DataWriteOptions): Promise<void>,

    // 根据当前聚焦或打开的文件(由 sourcePath)，应存放新文件的目录
    getNewFileParent(sourcePath: str, newFilePath?: str): TFolder,
    // 获取 创建新附件时 应存放该附件的目录
    getAvailablePathForAttachment(filename: str, sourcePath?: str):
Promise<str>
}
```

`TAbstractFile` 抽象类:

```
{
    vault: Vault,
    path: str,
    name: str,
    parent: TFolder | null
}
```

`TFolder < TAbstractFile` 类:

```
{
        children: TAbstractFile[],
        isRoot(): bool
}
```

`TFile < TAbstractFile` 类：

```
{
        stat: FileStats,
        basename: str,
        extension: str
}
```

MetadataCache

前置知识：`Linktext` 是 obsidian 的内链，由 `path` 和 `subpath` 组成，形如：`<path>#subpath`

`MetadataCache < Event` 类：

```
{
        // 获取 linkpath 的最佳匹配文件
        getFirstLinkpathDest(linkpath: str, sourcePath: str): TFile | null,
        // 获取文件的元数据
    getFileCache(file: TFile): CachedMetadata | null,
    // 获取文件的元数据
    getCache(path: str): CachedMetadata | null,
    // 获取文件的 linktext
    fileToLinktext(file: TFile, sourcePath: str, omitMdExtension?: bool):
str,

    // 映射表：<文件路径> => <出链文件路径> => <重边数量>
    resolvedLinks: Record<str, Record<str, number>>,
    // 映射表：<文件路径> => <无法解析的出链> => <重边数量>
    unresolvedLinks: Record<str, Record<str, number>>,

    on(name: 'changed', cb: (file: TFile, data: str, cache: CachedMetadata)
=> any, ctx?: any): EventRef,
    on(name: 'deleted', cb: (file: TFile, prevCache: CachedMetadata | null)
=> any, ctx?: any): EventRef,
    on(name: 'resolve', cb: (file: TFile) => any, ctx?: any): EventRef,
    on(name: 'resolved', cb: () => any, ctx?: any): EventRef
}
```

`CachedMetadata` 接口：

```
{
    links?: LinkCache[],
    embeds?: EmbedCache[];
    tags?: TagCache[];
    headings?: HeadingCache[];
    sections?: SectionCache[];
    listItems?: ListItemCache[];
    frontmatter?: FrontMatterCache;
    frontmatterPosition?: Pos;
    frontmatterLinks?: FrontmatterLinkCache[];
    blocks?: Record<string, BlockCache>;
}
```

Scope

`Scope` 类:

```
{
        constructor(parent?: Scope),
        // 注册一个事件
        // 注: key 字段需参考 ⇒ https://developer.mozilla.org/en-
US/docs/Web/API/UI_Events/Keyboard_event_key_values
        register(modifiers: Modifier[], key: str | null, func:
KeymapEventListener): KeymapEventHandler,
        // 通过句柄删除事件
        unregister(handler: KeymapEventHandler): void
}
```

keymap

`Keymap` 类:

```
{
        pushScope(scope: Scope): void,
        popScope(scope: Scope): void,
        static isModifier(evt: MouseEvent | TouchEvent | KeyboardEvent,
modifier: Modifier): bool,
        static isModEvent(evt?: UserEvent | null): PaneType | bool
}
```

编辑器

`obsidian` => `Editor`, `EditorPosition`, `EditorRange`, `EditorChange`
`obsidian` => `MarkdownView`

`Editor` 抽象类:

```
{
        getDoc(): this,
        refresh(): void,

        getValue(): str,
        setValue(content: str): void,
        lineCount(): num,
        getLine(line: num): str,
        setLine(n: num, text: str): void,
        lastLine(): num,

        somethingSelected(): bool,
        getRange(from: EditorPosition, to: EditorPosition): str,
        replaceSelection(replacement: str, origin?: str): void,
        replaceRange(replacement: str, from: EditorPosition, to?:
EditorPosition, origin?: str): void,
        getCursor(s?: 'from' | 'to' | 'head' | 'anchor'): EditorPosition,
        setCursor(pos: EditorPosition | num, ch?: num): void,

        getSelection(): str,
        setSelection(anchor: EditorPosition, head?: EditorPosition): void,
        setSelections(ranges: EditorSelectionOrCaret[], main?: num): void,
        listSelections(): EditorSelection[],

        // 聚焦/失焦 相关
        hasFocus(): bool,
        focus(): void,
        blur(): void,

        // 阅读位置相关
        getScrollInfo(): { top: num, left: num },
        scrollTo(x?: num | null, y?: num | null): void,
        scrollIntoView(range: EditorRange, center?: bool): void,

        undo(): void,
        redo(): void,
        exec(cmd: EditorCommandName): void,
        transaction(tx: EditorTransaction, origin?: str): void,
        wordAt(pos: EditorPosition): EditorRange | null,

        posToOffset(pos: EditorPosition): num,
        offsetToPos(offset: num): EditorPosition,

        processLines<T>(read: (line: num, lineText: str) => T | null, write:
(line: num, lineText: str, value: T | null) => EditorChange | void,
ignoreEmpty?: bool): void,
}
```

`EditorPosition` 接口：

```
{
        line: num,
        ch: num
}
```

`EditorRange` 接口：

```
{
        from: EditorPosition,
        to: EditorPosition
}
```

`EditorChange` 接口：

```
{
        text: str,
        from: EditorPosition,
        to?: EditorPosition
}
```

相关函数的应用

> 参考：Plugin

`MarkdownPostProcessor` 接口：

```
{
        (el: HTMLElement, ctx: MarkdownPostProcessorContext): Promise<any> |
void,
        sortOrder?: num
}
```

`MarkdownPostProcessorContext` 接口：

```
{
        docId: str,
    sourcePath: str,
    frontmatter: any | null | undefined,
    addChild(child: MarkdownRenderChild): void,
    getSectionInfo(el: HTMLElement): MarkdownSectionInformation | null
}
```

例子：（如：`example` => `[EXAMPLE]`）

```
this.addCommand({
        id: '',
        name: '',
        editorCallback: (editor, ctx)=>{
                editor.replaceRange(editor.getSelection().toUpperCase(),
editor.getCursor('from'), editor.getCursor('to'))
                editor.replaceRange(`[`, editor.getCursor('from'))
                editor.replaceRange(`]`, editor.getCursor('to'))
        }
})
```

例子2 => 将 csv 代码块实时渲染为表格：

```
this.registerMarkdownCodeBlockProcessor('csv', (source, el, ctx)=>{
        let table = el.createEl('table')
        let tbody = table.createEl('tbody')

        source.split('\n')
                .filter((row) => row.length>0)
                .forEach((v)=>{
                        let tr = tbody.createEl('tr')
                        v.split(',').forEach((w)=>{
                                tr.createEl('td', {text: w})
                                // tr.createEl('td', {text: w}, (el)=>{
                                //      let c =
[0,1,2,3,4,5,6,7,8,9,'a','b','c','d','e']
                                //      let rd =
()=>c[Math.floor(Math.random()*16)]
                                //      el.setAttr('style', `color:
#${rd()}${rd()}${rd()}${rd()}${rd()}${rd()}; text-align: center`)
                                })
                        })
                })
        // table.setAttr('style', 'width: 95%')
})
```

编辑器扩展

> 前置知识：js-tools-main > CodeMirror

`@codemirror/view` => `PluginValue`, `ViewPlugin`, `ViewUpdate`, `EditorView`

```
import {ViewUpdate, ViewPlugin, PluginValue, EditorView} from
'@codemirror/view'
class MyPlugin implements PluginValue{
        constructor(view: EditorView) {

        }
        update(update: ViewUpdate) {

        }

        destroy() {

        }
}

export const myPlugin = ViewPlugin.fromClass(MyPlugin)
```

实例

显示 leaf 的多种方式：

```
// (1)
let f = this.app.vault.getFileByPath('readme.md')
if (f){
        let leaf = this.app.workspace.getLeaf(true)
        await leaf.openFile(f)
        // this.app.workspace.setActiveLeaf(leaf, {focus: false})
}

// (2) 打开自定义 ItemView
let leaf1 = this.app.workspace.getLeaf(true)
await leaf1.open(new MyItemView(leaf1,this))
// this.app.workspace.setActiveLeaf(leaf1, {focus: false})

// (3)
await this.app.workspace.openLinkText('obsidian#社区插件', '', true)
```