插件系统

插件系统有着非常的强大的功能，请不要安装来源不明、加密、代码复杂难以审计的插件，它可能会损坏你的电脑。

插件系统能够做什么?

- 修改你的 APP 主题、语言，管理你的配置、订阅、规则集、内核。
- 对生成的配置进行修改、对订阅的结果进行修改。
- 集成第三方程序，扩展 GUI 的能力。
- GUI 中的一切操作均可通过插件进行。

插件运行原理

本 GUI 中的插件是一系列的触发器，所谓的触发器就是在满足某个条件时自动执行，GUI 支持了下面几种触发器：

- `手动触发`：点击运行按钮时会被触发，GUI 会执行源码中的 `onRun` 方法。
- `更新订阅时`：更新订阅时会被触发，GUI 会执行源码中的 `onSubscribe` 方法，并传递一个参数，其值是节点列表数组。此方法需要返回一个节点列表的数组。
- `生成配置时`：生成配置文件时会被触发，GUI 会执行源码中的 `onGenerate` 方法，并传递一个参数，其值是一个对象，里面包含了 core 的配置，此方法需要将处理后的参数返回，或原样返回。
- `启动 APP 时`：启动 APP 时会被触发，GUI 会执行源码中的 `onStartup` 方法，没有传递参数，此方法无需返回任何值。
- `关闭 APP 时`：关闭 APP 时会被触发，GUI 会执行源码中的 `onShutdown` 方法，没有传递参数，此方法无需返回任何值。

插件开启了 `需要安装` 参数时，界面会多出 `安装` 和 `卸载` 按钮，点击后 GUI 会执行源码中的 `onInstall` 和 `onUninstall` 方法。可用来做插件的初始化工作与善后工作，当 `onInstall` 方法执行没有出错，GUI 会认为插件执行安装成功，将插件标记为已安装，当 `onUninstall` 方法执行没有出错，GUI 会认为插件执行卸载成功，将插件标记为已卸载（即未安装）。

插件配置了 `菜单` 时，右键插件卡片会出现对应的菜单项，点击后会执行相应的方法。

插件配置了 `配置` 时，右键插件【配置插件】，可对插件进行配置。

插件状态码

插件钩子方法中可返回状态码，例如 onRun 方法中返回状态码 `1` 表示此插件启动完成，正在运行中，再例如自定义菜单项 `Stop` 方法中返回状态码 `2` ，表示此插件已经停止运行，正常退出了。

状态码如下：

- 0 无状态，推荐 onInstall 和 onUninstall 方法中作为返回值

- 1 运行中，推荐 onRun 方法中作为返回值
- 2 已停止，推荐自定义菜单 Stop 方法中作为返回值

以下是一个示例，包含了所有的钩子方法。

javascript

```
/**
 * 插件钩子：运行按钮 - onRun
 */
const onRun = async () => {
  await StartMyProgram();
  return 1; // 表示插件正在运行中
};

/**
 * 自定义菜单项：停止 - Stop
 */
const Stop = async () => {
  await StopMyProgram();
  return 2; // 表示已经停止运行
};

/**
 * 自定义菜单项：运行 - Start
 */
const Start = async () => {
  await StartMyProgram();
  return 1; // 表示插件正在运行中
};

/**
 * 插件钩子：安装按钮 - onInstall
 */
const onInstall = async () => {
  await InstallMyProgram();
  return 0; // 表示初始状态
};

/**
 * 插件钩子：卸载按钮 - onUninstall
 */
const onUninstall = async () => {
  await UninstallMyProgram();
  return 0; // 表示初始状态
```

```
};

/**
 * 插件钩子：更新订阅时
 */
const onSubscribe = async (proxies, subscription) => {
  return proxies;
};

/**
 * 插件钩子：生成配置时
 */
const onGenerate = async (config, profile) => {
  return config;
};

/**
 * 插件钩子：启动APP时
 */
const onStartup = async () => {};

/**
 * 插件钩子：关闭APP时
 */
const onShutdown = async () => {};

/**
 * 插件钩子：APP就绪后
 */
const onReady = async () => {};

/**
 * 插件钩子：计划任务执行时
 */
const onTask = async () => {};

/**
 * 插件钩子：配置插件时
 */
const onConfigure = async (config, old) => {};
```

插件编写规范

1、代码应格式化、易于阅读、不能加密；

2、在程序 data 目录下进行 IO 操作，不访问用户私有目录；

3、临时文件应存放在 data/.cache 目录，用完需要删除对应的文件；

4、第三方程序应放置到 data/third 目录，卸载时需要删除对应目录；

5、禁止动态创建 script、style 等标签，引入外部 js、css 等操作；

6、对系统有侵入性的修改，需要在卸载时进行恢复操作。

插件编写示例

1、手动触发插件示例

首先创建一个插件：

然后编写对应的代码：

最后可以尝试安装、运行、卸载插件

下面的插件示例需要同样的操作，先创建插件、再编写插件代码、最后运行。

2、更新订阅时插件代码示例

javascript

```javascript
// params: proxies是一个代理数组
// params: metadata是订阅元数据
// return: 请返回一个代理数组[]
const onSubscribe = (proxies, metadata) => {
  // 示例：把节点名称中的新加坡替换为空
  proxies = proxies.map((v) => {
    return {
      ...v,
      name: v.name.replace("新加坡", ""),
    };
  });
  return proxies;
};
```

3、生成配置时插件代码示例

javascript

```javascript
// params: config是已生成的标准的内核配置，即config.yaml文件的内容
// params: metadata是生成内核配置的源数据，即GUI所使用的profile数据
// return: 请返回标准的内核配置
const onGenerate = (config, metadata) => {
  if (metadata.name == "某个profile") {
    // 仅当某个profile时，才处理
    // 一些处理...
  }
  // 移除tun配置
  delete config.tun;
  // 关闭DNS服务器
  config.dns.enable = false;
  return config;
};
```

4、启动 APP 时插件代码示例

javaScript

```javascript
const onStartup = () => {
    alert('APP启动了')
}
```

5、关闭 APP 时插件代码示例

javaScript

```
const onShutdown = () => {
    alert('APP关闭了')
}
```

插件能力：Plugins

在上面我们演示了 Plugins.message、Plugins.HttpGet，那么插件对象 Plugins 还有哪些能力呢，你可以在软件界面按下 Ctrl+Shift+F12 打开开发者面板，切换到控制台，输入 Plugins 并回车查看，具体的使用示例可以看源码。

更多的示例

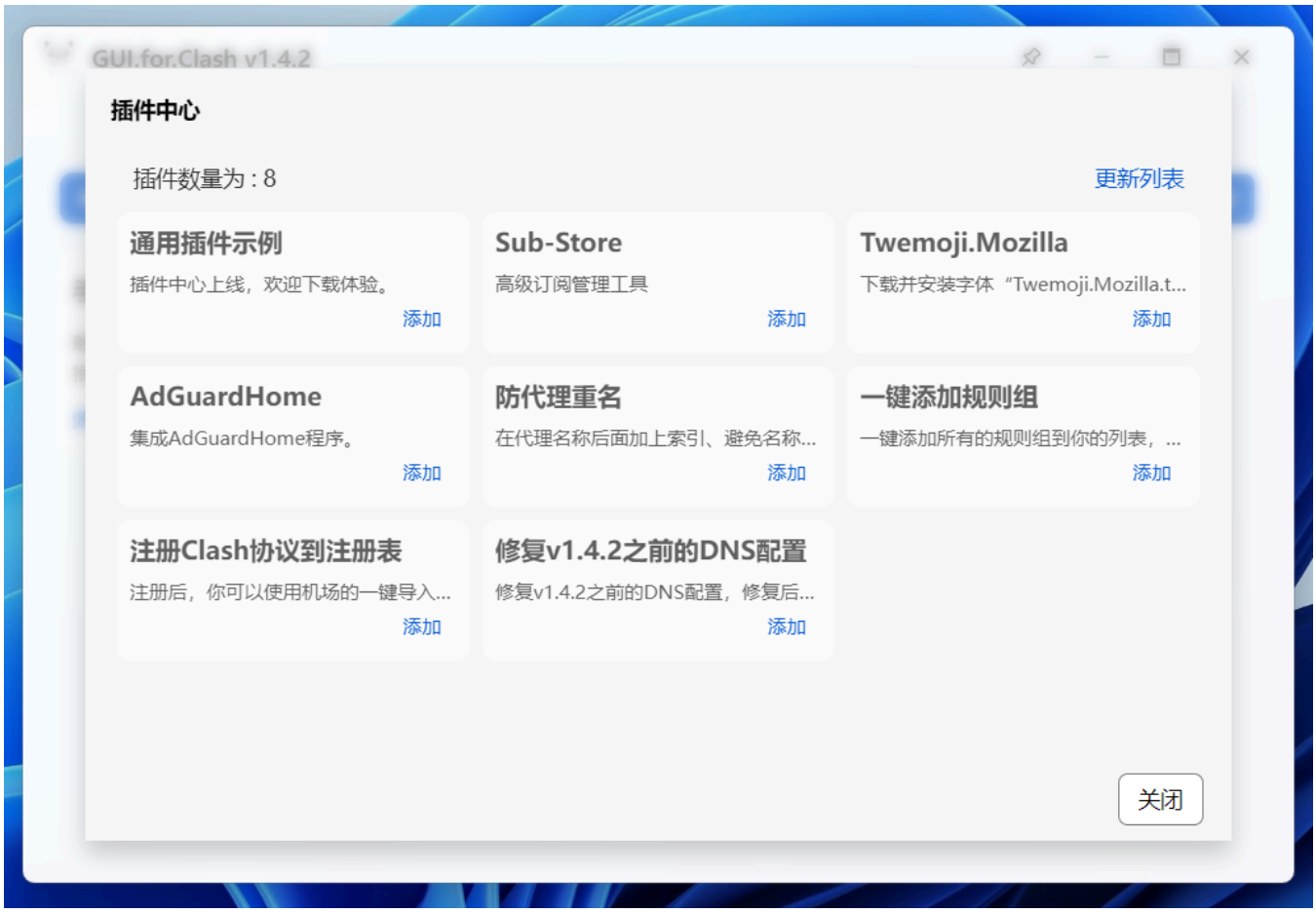
javascript

```
// 消息提示示例
const { id } = Plugins.message.info('GUI.for.Cores', 4_000)
await Plugins.sleep(1_000)
Plugins.message.update(id, 'is')
await Plugins.sleep(1_000)
Plugins.message.update(id, 'powerful')
await Plugins.sleep(1_000)
Plugins.message.destroy(id)

// APP设置示例
const appSettings = Plugins.useAppSettingsStore()
appSettings.app.theme = 'dark' // light
appSettings.app.lang = 'en' // zh

// 系统代理管理示例
const envStore = Plugins.useEnvStore()
envStore.setSystemProxy()
envStore.clearSystemProxy()
envStore.switchSystemProxy()

// 内核管理示例
const kernelApiStore = Plugins.useKernelApiStore()
kernelApiStore.startKernel()
kernelApiStore.stopKernel()
```

```
kernelApiStore.restartKernel()

// 配置管理示例
const profilesStore = Plugins.useProfilesStore()
profilesStore.addProfile(p: ProfileType)
profilesStore.editProfile(id: string, p: ProfileType)
profilesStore.deleteProfile(id: string)

// 订阅管理示例
const subscribesStore = Plugins.useSubscribesStore()
subscribesStore.addSubscribe(s: SubscribeType)
subscribesStore.editSubscribe(id: string, s: SubscribeType)
subscribesStore.deleteSubscribe(id: string)
subscribesStore.updateSubscribe(id: string)

// 规则组管理示例
const rulesetsStore = Plugins.useRulesetsStore()
rulesetsStore.addRuleset(r: RuleSetType)
rulesetsStore.editRuleset(id: string, r: RuleSetType)
rulesetsStore.deleteRuleset(id: string)
rulesetsStore.updateRuleset(id: string)

// 插件管理示例
const pluginsStore = Plugins.usePluginsStore()
pluginsStore.addPlugin(p: PluginType)
pluginsStore.editPlugin(id: string, p: PluginType)
pluginsStore.deletePlugin(id: string)
pluginsStore.updatePlugin(id: string)
pluginsStore.reloadPlugin(plugin: PluginType, code = '')
pluginsStore.updatePluginTrigger(plugin: PluginType)

// 计划任务管理示例
const scheduledTasksStore = Plugins.useScheduledTasksStore()
scheduledTasksStore.deleteScheduledTask(id: string)
scheduledTasksStore.editScheduledTask(id: string, s: ScheduledTaskType)
scheduledTasksStore.addScheduledTask(s: ScheduledTaskType)
```

插件中心、注意事项

插件中心是为了方便用户下载常用的插件而设立的仓库，其源码可在此仓库查看：Plugin-Hub。



从插件中心添加的插件不建议修改元数据，也就是插件卡片右上角的【编辑】按钮(已改为 【开发】)，因为发布到插件中心的插件都是调试好的，该有哪些触发器就有哪些触发器，该有哪些菜单以及配置项也都是设计好的，有些会用户随意编辑这些插件，比如添加了源码里没有实现的触发器，就会导致插件执行失败。

那么为什么不限制编辑按钮呢？因为我们想把最大的权限交给用户，毕竟有些有想法的用户想对已有插件进行扩展，增加自己想要的功能，GUI 不会限制这部分用户。

如果编辑了乱了插件元数据该怎么办？那就卸载、删除插件，然后来到插件中心重新添加、安装。

为什么有些插件有安装卸载按钮，有些插件没有？这个是根据每个插件的工作原理来的，例如 AdHuardHome 插件，它本身没有任何功能，依赖第三方程序，所以就需要安装与卸载按钮来下载程序与删除程序。再例如节点转换插件，它不依赖任何第三方程序，自然就不需要安装卸载了。

有些插件设计时会提供一些配置项供用户填写，右键插件卡片的第三个菜单项【配置插件】就能打开对应页面，有些插件没有这一项，说明插件不需要用户配置。

插件右上角的更新按钮仅用做更新插件源码，而不是更新插件元数据，所以如果插件更新了元数据，比如添加了一个菜单项或一个配置项，就需要删除插件重新添加。在此之前别忘记了先更新插件列表。

注意事项大概就这么多，最后欢迎各位为 GUI 编写插件，并提交你的插件到插件中心。

Last updated: 2024/12/22 17:17

Previous page
运行原理

Next page
计划任务系统