

# RT-Thread Settings Web化需求分析

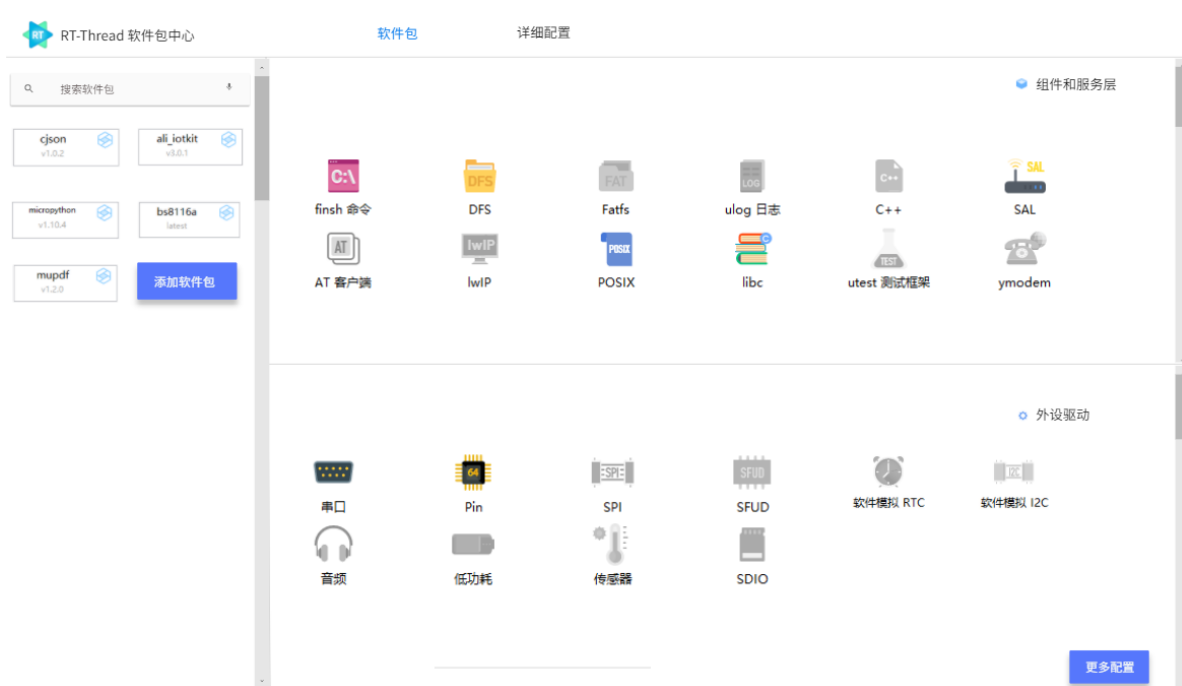
## 前端

### 技术栈


- VUE框架
- JavaScript + HTML + CSS

### 功能需求

#### (1)【软件包添加】界面



- 添加软件包


- 点击添加软件包 ，此按钮随着软件包的数量动态向下移动，与目前 studio 中逻辑一样，点击后跳转到下图所示页面，添加成功后在左侧显示刚添加的软件包信息：



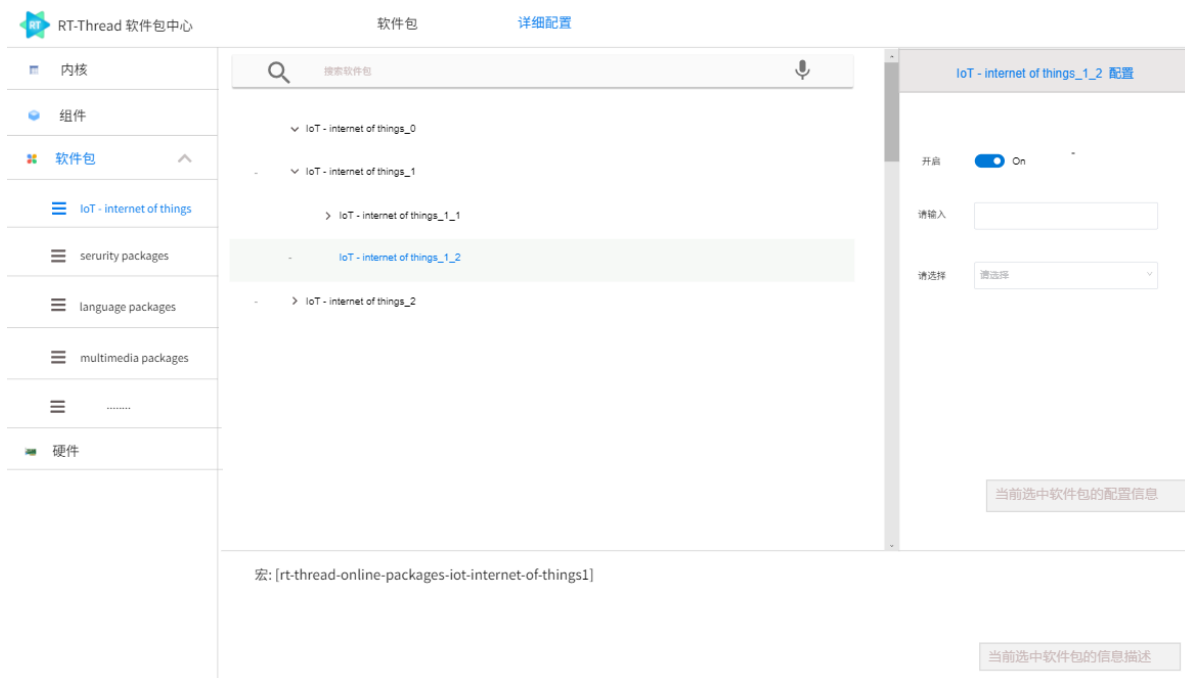
- 搜索软件包

- 在搜索框输入 软件包 名称，自动过滤出符合条件的软件包显，并且支持模糊搜索



- 软件包定位
  - 点击某个软件包后，自动跳转到【详细配置】界面（在下一节查看），并定位到对应的软件包在树结构中的位置，即选中该软件包，并在【详细配置】界面右侧显示配置信息
- 使能 组件和服务层/外设驱动
  - 组件和服务层/外设驱动 都是一些常用的配置
  - 在此 点击 组件和服务层/外设驱动 某个图标时，根据使能的实际情况，对图片进行变亮/变灰的操作；根据具体组件和服务层/外设驱动的信息，组装使能的 json 数据，通过约定的接口 notifyStatusChanged 发送 json 数据到后端，由后端进行处理
- 更多配置
  - 点击更多配置 ，自动跳转到【详细配置】界面

## (2) 【详细配置】界面



- 配置分类
  - 包括内核、组件、软件包、硬件、示例，根据实际情况，不一定需要显示配置全部分类
- 软件包显示
  - 解析后端发送过来的 .rtmenus，将所有分类显示在左侧菜单中
  - 当点击某个菜单时，在中间显示该分类的所有子配置项（以树形结构展示）

- 由于软件包下也有明确的分类，软件包数据的第一层级显示在软件包的子菜单中，从第二层级开始在中间显示配置项；其他 内核、组件等菜单没有子菜单，直接在中间按层级显示所有配置项
  - 在右下侧展示当前选中软件包的描述信息
- 软件包搜索
  - 在搜索框输入 软件包 名称，自动过跳转并定位到该软件包在树形结构上的位置
- 软件包配置
  - 当某个软件包或者子节点被选中时，在右侧显示该软件包的配置项
  - 配置完成后，解析与后端交互发来的数据，自动刷新当前的树形结构

## 后端

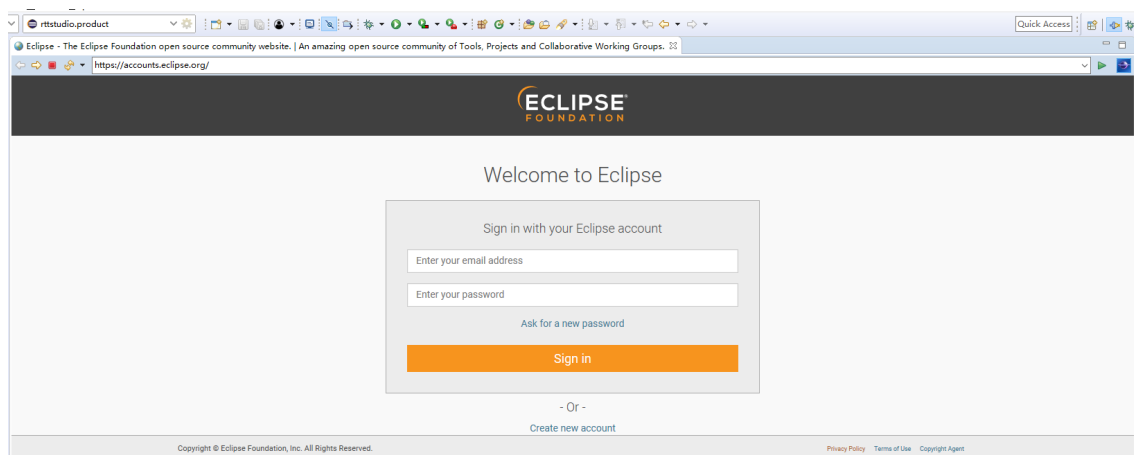
### 技术栈


- Eclipse插件框架
- Java
- Python ( k2j, kcs )

### 功能需求

- 启动 Web 端 RT-Thread Settings

将 chrome 内核浏览器 (org.eclipse.swt.chromium.Browser) 置于编辑器 (org.eclipse.ui.part.MultiPageEditorPart) 中，以实现浏览器直接嵌套在 Studio 中, 类似于 Eclipse 的账户登录功能的实现。



- 打开 RT-Thread Settings
  - 双击  **RT-Thread Settings**
  - 打开扩展点 org.eclipse.ui.editors 下 id 为 org.rt-thread.studio.rttconfig.ui.editor 的编辑器
  - 在此编辑器中，新建一个 chrome 内核浏览器
  - 将此浏览器绑定一个与前端约定的接口 packageCenter，传递网址数据
- 发送软件包配置的数据给前端
  - 获取 k2j 本地路径
  - 执行 k2j.exe --config ./config --kconfig ./Kconfig --output json\_menus ./settings/.rtmenus，生成 .rtmenus
  - 将 .rtmenus 发送给前端，约定接口 rtmenusData
- 启动 kcs 服务
  - 获取 kcs 本地路径
  - 设置 python 环境和本地软件包路径，执行 kcs.exe, --config, ./config, --kconfig, Kconfig，开启 kcs 服务
  - 开启 kcs 监听

- 监听 Web 端的修改
  - 与前端约定接口 notifyStatusChanged 来监听前端配置的修改
  - 前端配置发生变化
    - 生成 kcs 能够解析的 json 数据
    - 在 JS 中将 json 数据传递到 notifyStatusChanged 中
  - 后端监听到前端配置变化
    - 通过 notifyStatusChanged 接收 json 数据
    - 将 json 数据在 kcs 中解析
    - 将 kcs 返回的数据通过 notifyStatusChanged 发送给前端
    - 将发生变化的数据存储到 Map 中，用于在关闭时调用 kcs 对工程进行配置
    - 开启脏检查
  - 前端刷新UI
    - 前端获取 notifyStatusChanged 返回的数据
    - 解析返回的数据，配置 UI 界面
- 关闭 Web 端 RT-Thread Settings
  - kcs 服务执行 Map 中的数据
  - 关闭脏检查
  - 关闭 kcs 服务
  - 检查是否开启 C++ 功能
  - 配置工程，更新软件包

## 前后端通信

- 接口定义
  - packageCenter：开启 RT-Thread Settings
  - rtmenusData：后端传递前端需要配置的数据
  - notifyStatusChanged：监听前端配置的变化

- 服务和请求

采用 HTTP GET请求

- 前端数据交互（示例）

```
$(".add").click(function(){
    ...
    var jsonData = `{"pkgName":"${params}", "enable" :
"${paramsEnable}"}`; // jsonData: 前端发送的数据
    // result: 前端接收的数据
    result = updatePackages(jsonData); // updatePackages 为与后端约定的
接口
})
```

- 后端 ( Studio ) 数据交互（示例）

```
new CustomFunction(browser, "updatePackages"); // updatePackages 为与前端约定的接口
```

```
class CustomFunction extends BrowserFunction {  
    CustomFunction(Browser browser, String name) {  
        super(browser, name);  
    }  
    public Object function(Object[] arguments) { // arguments: 后端接收的  
数据  
        ... // 逻辑代码  
        return data; // data: 后端发送的数据  
    }  
}
```