前言

我申请的项目是wasm函数扩展。这个项目的目标是设计并实现 wasm 扩展框架，可以方便用户使用 WASM 函数扩展 ekuiper 的处理能力。这个项目选择 wasmedge 作为沙箱执行引擎，使用 wasmedge-go SDK 加载，使用wasm文件，以达到函数调用的目地。

WebAssembly是一种运行在现代网络浏览器中的新型代码，并且提供新的性能特性和效果。而且，你在不知道如何编写WebAssembly代码的情况下就可以使用它。根据它的特性，只需用户提供相应函数的wasm文件，就可以使用 wasmedge-go SDK 与文件交互，获取函数运行结果。

整体框架

项目执行的整体流程大致是：用户使用任意不计的语言编写一个函数，将其编译成 .wasm文件，添加到ekuiper可以索引到的宿主机下面，然后通过插件信息告诉ekuiper，在用户方看来，用户就可以直接调用函数进行使用了。而在后台，ekuiper启动沙盒环境，根据配置文件找到加载对应的 wasm文件，从wasm 文件中获取信息，与其进行参数传递与调用。响应用户的函数使用信息。要保证ekuiper 能够灵活的调用所有函数，需要设计一个wasm 函数扩展框架来管理后台事务。扩展框架获取从 ekuiper 发过来的配置信息，根据配置信息找到并加载用户提供的 wasm 文件，使用 wasmedge 执行引擎启动沙盒，准备就绪即可发出信息表示已经准备好了。这时在前台，用户就可以使用函数，如需传参与返回值，也可与沙箱进行交互了。

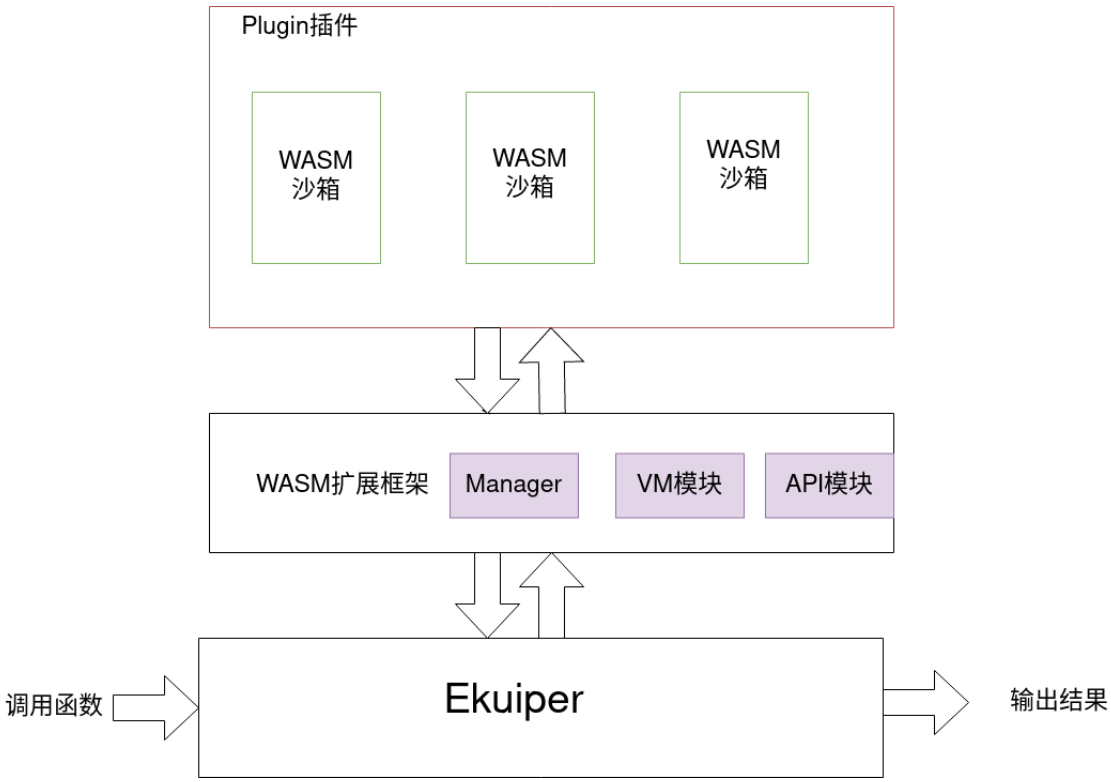
在上文描述的大致流程中，三方通过插件配置信息进行各自的部分，详细描述如下：

1. ekuiper 给 wasm 扩展框架发送配置信息，告诉框架插件名称，需要的沙箱执行引擎，用户提供的 wasm 文件所在位置，以便后续的加载可以找到，使用的函数名称等。这些初始信息汇总的配置文件结构可以设计如下：

```
{
  "name": "car",                    //插件名称
  "instance_num": 4,                //实例数量
  "vm_config": [
      "engine" : "wasmedge",        //执行引擎
      "path" : "/plugin/XX.wasm",   //wasm 文件路径
  ],
  "functions": [
    "link", "rank"                  //所需函数
  ]
}
```

2. wasm扩展框架获取配置信息，对未来可能存在多个的配置信息进行管理，管理操作包括增加配置，删除，修改，全局检查等。也负责接收用户通过 ekuiper发送过来的函数调用信息，与配置文件所对应的沙箱进行传参与获取返回值。
3. Plugin 插件中是一系列的 wasm 沙箱的集合 ，这些沙箱使用 wasmedge 执行引擎运行，接收 wasm 扩展框架传来的函数参数信息，并将 获取到的 wasm文件传出的返回值告知 wasm 扩展框架，完成值的传递。

根据上述细分功能，可以构建整体框架如下：



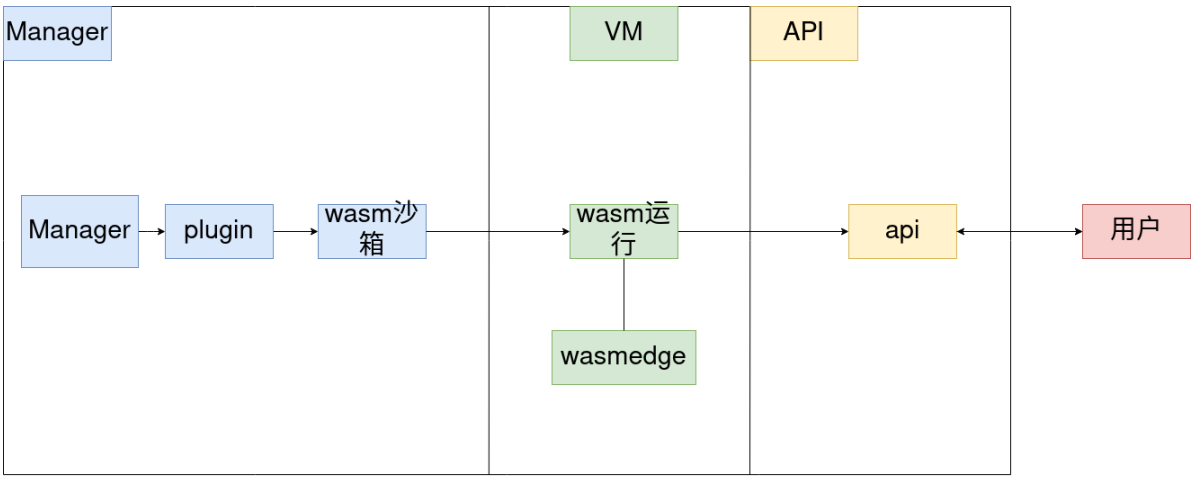
扩展框架

从上述整体框架的描述以及设计图来看，扩展框架作为中间件，在整个流程中起到承上启下且作为信息传递的通道的重要作用。更重要的是，它需要对未来可能添加的多个插件配置进行管理，以帮助各个插件互不影响的运行，也需要对配置文件进行管理，使整体结构清晰有序，方便用户对插件进行管理。

当 ekuiper 加载上述配置的时候，用户通过 wasm 扩展框架获取指定插件的沙箱实例，然后后续与 wasm 沙箱暴露的 api 进行交互，调用函数。

根据框架所需要的功能，wasm 扩展框架大致分为三个部分，分别是Manager管理、VM执行 和 API 交互模块三个部分，他们分别负责管理，运行与交互三部分功能，如下图：

1. Manaager管理模块：负责对多个插件和沙箱的管理，基础管理功能需要有：添加用户配置方便后续进行管理，包括增加，删除以及修改等。
2. VM 运行模块： 使用 wasmedge 执行引擎运行沙箱，在这个模块与 wasm 沙箱进行交互，在用户通过 api 传来参数时，可以使沙箱接收数据并传回值。
3. API 模块： 提供对外接口，将值传给 VM 模块。



函数调用与传值

以上，都是描述了具体的过程，在这些过程中，函数传值与传回值占了很重要的部分，以官网(https://wasmedge.org/book/en/embed/go/function.html)提供的原始的(将要编译成wasm文件的) rust 文件为例。

在 [Rust 项目]中使用 (https://github.com/second-state/WasmEdge-go-examples/tree/master/wasmedge-bindgen/go_BindgenFuncs/rust_bindgen_funcs) `[wasmedge_bindgen]`宏。 这些带注释的函数将由 Rust 编译器自动检测并转换为 WebAssembly 函数，现在就可以从 `wasmedge-bindgen`使用go SDK ， 例如：

```
#[wasmedge_bindgen]
pub fn create_line(p1: String, p2: String, desc: String) -> Result<Vec<u8>,
String> {
  let point1: Point = serde_json::from_str(p1.as_str()).unwrap();
  let point2: Point = serde_json::from_str(p2.as_str()).unwrap();
  let length = ((point1.x - point2.x) * (point1.x - point2.x) + (point1.y -
point2.y) * (point1.y - point2.y)).sqrt();

  let valid = if length == 0.0 { false } else { true };

  let line = Line { points: vec![point1, point2], valid: valid, length: length,
desc: desc };

  return Ok(serde_json::to_vec(&line).unwrap());
}
```

在rust中使用了[wasmedge_bindgen],那么便可以使用 execute 来使用这个函数。

```
res, err := bg.Execute("create_line", "{\"x\":2.5,\"y\":7.8}", "
{\"x\":2.5,\"y\":5.8}", "A thin red line")
  if err == nil {
    fmt.Println("Run bindgen -- create_line:", string(res))
  } else {
    fmt.Println("Run bindgen -- create_line FAILED", err)
  }
```

得到结果：

```
Run bindgen -- create_line: {"points":[{"x":1.5,"y":3.8},
{"x":2.5,"y":5.8}],"valid":true,"length":2.2360682,"desc":"A thin red line"}
```

当然，上面使用的是rust语言写原始文件的，但不止rust语言，go语言也可以被编译成wasm文件，且能执行，例如，这是一个简单的打印函数，将其编译成wasm文件，可以在浏览器，nodejs上运行，得到结果，并且也可以使用go语言执行，只需要写一个落地函数main.go，调用模块即可。

```go
package main
import (
    "github.com/common-nighthawk/go-figure"
)
func main() {}
//export HelloWorld
func HelloWorld() {
    myFigure := figure.NewFigure("Hello World", "", true)
    myFigure.Print()
}
```

上述我列举的无论是 rust语言，还是go语言写的原始函数文件，最后都可以使用 go SDK 接口进行调用，实现使用函数的功能。

时间安排

我根据项目和自身的实际情况，时间安排如下：

时 间	项目进度
6 月 16 日—— 6 月 30 日	了解wasmedge-go相关实例，学习官网上的函数使用
7 月 1 日 —— 7 月 10 日	编写 wasm 扩展框架架构，Manager，VM 和 API 模块内的函数命名，参数设置等
7 月 11 日 —— 8 月 5 日	完成 Manager ，VM ， API 模块函数的实现，并撰写项目文档，准备项目中期报告
8 月 6 日 —— 8 月 10 日	编写相应的单元测试，修改 Bug ， 完善功能
8 月 11 日 —— 9月 25日	将 扩展框架 集成到 ekuiper 中
9 月 25日 —— 9 月 30日	继续改进，撰写说明文档，总结项目最终报告

https://vscode.cdn.azure.cn/stable/c3511e6c69bb39013c4a4b7b9566ec1ca73fc4d5/code_1.67.2-1652812855_amd64.deb