WebAssembly 组件模型与模块

于桐 2023.3.7

前言

- 如果大家写过Wasm程序的话可能对Wasm模块有一定了解。
- 一个Wasm模块是一个包含了一堆东西的Wasm程序文件，有文本（WAT）和二进制两种形式（两种形式等价，可以互相转换）
- Wasm模块的文件里大概包括：
- 导入声明（声明这个模块需要从外部导入哪些函数之类的东西）
- 导出声明（声明这个模块导出了哪些函数之类的给其他模块，或者host来用）
- 函数的代码
- 以及其他一些辅助的东西，在这里不太重要，所以不细讲了

Wasm模块的不足 – 复杂数据类型

- 已知：Wasm函数的参数和返回值类型只能是整数和浮点数
- 要传字符串/结构体/数组怎么办？
- 在同一个模块内，编译器可以有一套自己的ABI，只要模块内能工作就可以。
- 跨模块怎么办？没办法，没有统一的标准。一般我们采取把指针转成整数的方式来传递，但没有什么统一的标准。

- 组件模型为传递复杂的数据类型提供了标准化的接口。

Wasm模块的不足 – 接口

- 根据上文我们说的，Wasm模块可以指定这个模块需要导入的函数。
- 但只有函数是不够的，函数是无状态的，单纯的函数也不便于封装。
- 如果我们想导入一个实现了一组特定函数的对象怎么办？（interface）

- 组件模型给我们提供了interface。Interface指定一组函数签名，Wasm程序可以导出或者导入一个实现了特定interface的对象，使用interface的一方不需要关心interface的具体实现。

Wasm组件的目标

- https://github.com/WebAssembly/component-model/blob/main/design/high-level/Goals.md#component-model-high-level-goals
- 支持不同语言构建出来组件通过同一种ABI相互调用（可以传字符串、对象之类的东西，而模块只能传整数和浮点数）
- 支持可移植的、语言无关的、安全的对象接口定义
- 进一步增强Wasm的语言独立性、可扩展性、AOT友好性以及与Web平台的集成
- 支持渐进式的改进Wasm组件

Wasm组件解决了什么问题

- 根据上面我们说的，Wasm组件至少解决这些问题：
- 不同的Wasm组件之间传递复杂数据类型
- 支持导出语言独立的对象（满足一组由特定函数构成的接口）

- 更详细的可以见https://github.com/WebAssembly/component-model/blob/main/design/high-level/UseCases.md

Wasm组件的接口描述文件

- 既然Wasm组件给我们提供了导入、导出“接口”这种东西的功能，所以自然就有了一种用来描述这种接口的文档格式，被称为WIT
- Wasm Interface Type
- https://github.com/WebAssembly/component-model/blob/main/design/mvp/WIT.md

```
world my-world {
  import host: interface {
    log: func(param: string)
  }

  export run: func()
}
```

WIT

- 总的来说，一个WIT文件里描述了一个Wasm组件导入的导出以及一些相关的东西，这些东西包括：
- 导入和导出的函数
- 导入和导出的接口
- 上述函数和接口的参数或返回值所用到的类型，包括record（类似于结构体）、variant（类似于Rust的enum，一种tagged union）、union（类似于C的union，untagged union）、enum（类似于C的enum，映射到整数）等。

- 关于类型的更详细的信息可见 https://github.com/WebAssembly/component-model/blob/main/design/mvp/WIT.md#wit-types

WIT

- 来看一个复杂一点的WIT的例子

```
world my-world {
  import host: interface {
    log: func(param: string)
  }

  export run: func()
}
```

- 定义了一个被称为my-world的world（可以认为，一个world用来描述一个Wasm组件）
- 这个组件内导出一个被称为run的函数，这个函数不接受参数也没有返回值。
- 这个组件需要导入一个名字叫做host的对象，这个对象需要满足给定的接口。即，这个对象需要提供一个名为log的函数，接受一个string的类型参数，没有返回值。

组件模型的应用

- 既然组件模型这么好，又有WIT这么好的东西来做接口描述，那么我们怎么用呢？
- ~~*显然可以用来简化Wasm组件程序的开发*~~
- BytecodeAlliance的人做了 https://github.com/bytecodealliance/wit-bindgen，可以：
- 根据我们写的WIT文件，帮我们生成处理interface、字符串、数组、结构体之类的复杂类型的代码。
- 我们只需要写具体函数的实现即可。
- 虽然现在没有什么能直接输出Wasm组件的编译器，但可以在把我们写的函数实现和wit-bindgen生成的东西链接到一起后现输出一个Wasm模块，然后可以使用Bytecodealliance提供的工具链把模块变成Wasm组件。

wit-bindgen + C

```
wit-bindgen c .\test.wit -w "test.my-world"
```

- 将前几张ppt里展示的那个WIT文件保存为test.wit，然后使用wit-bindgen生成一下。
- -w参数用以指定我们正在使用的world，格式为<WIT文件名>.<world>名。
- 然后wit-bindgen会给我们生成一个my_world_component_type.o文件，一个my_world.c文件，一个my_world.h文件。
- 我们需要在构建的时候把my_world_component_type.o链接进去，这个文件里包含了几个工具section，用来给后续工具链里的模块转组件的工具用。
- my_world.c里包含了一些工具函数的实现，比如操作组件模型中的字符串类型的工具函数
- my_world.h中包含了我们写的组件从外部导入的函数的声明（host接口下的log函数），以及我们需要自己写的函数的声明（run函数）

my_world.h

```c
typedef struct {
  char*ptr;
  size_t len;
} my_world_string_t;

// Imported Functions from `host`
void host_log(my_world_string_t *param);

// Exported Functions from `my-world`
void my_world_run(void);

// Helper Functions

void my_world_string_set(my_world_string_t *ret, const char*s);
void my_world_string_dup(my_world_string_t *ret, const char*s);
void my_world_string_free(my_world_string_t *ret);
```

一个更好的例子

- https://github.com/eunomia-bpf/c-rust-component-test
- 我们在这个例子中演示了通过wit-bindgen分别用C和Rust来编写组件，并且让他们互相调用，然后在wasmtime这个支持组件模型的Wasm运行时上跑的例子。

另一个例子

- 我们做了一个用Rust编写eBPF用户态程序，并使用btf2wit生成内核结构体的WIT定义随后解析数据的例子，具体可以见 https://github.com/eunomia-bpf/wasm-bpf/tree/main/examples/rust-bootstrap
- 这个例子仍然不完善，但足以演示btf2wit的用途。

关于WIT和组件模型的更多信息

- 组件模型现在还很不成熟，目前为止没有任何一个完整支持组件模型的Wasm运行时（https://github.com/bytecodealliance/wasm-tools/tree/main/crates/wasm-compose#implementation-status）。
- 但wasmtime现在对组件模型提供初步支持（虽然一堆bug），也因此上面的例子用的是wasmtime。

- 关于WASI。现在已经有了组件模型使用WASI的标准（preview2，https://github.com/bytecodealliance/preview2-prototyping），但目前为止没有任何运行时实现了这个标准。所以暂时组件模型里用不了WASI。

- 关于更多Wasm组件相关的工具。BytecodeAlliance开发了wasm-tools(https://github.com/bytecodealliance/wasm-tools)，提供了从模块创建组件、合并多个组件等功能。

eunomia-bpf 做的一些关于组件模型的工作

- https://github.com/eunomia-bpf/c-rust-component-test
- 一个让不同语言写的Wasm组件协同工作的例子

- https://github.com/eunomia-bpf/btf2wit
- 一个从BTF（BPF Type Format，一种让eBPF程序用起来更舒服的调试信息格式）生成WIT的小工具。

参考链接

- https://github.com/WebAssembly/component-model/blob/main/design/high-level/Goals.md
- https://github.com/WebAssembly/component-model/blob/main/design/high-level/UseCases.md
- https://github.com/WebAssembly/component-model/blob/main/design/high-level/Choices.md
- https://github.com/WebAssembly/component-model/blob/main/design/high-level/FAQ.md