插入

一个 WebAssembly 插件。

Typst 能够与编译为 WebAssembly 的插件交互。插件函数可以接受多个字节缓冲区作为参数并返回单个字节缓冲区。它们通常应该包装在惯用的 Typst 函数中，这些函数在本机 Typst 类型和字节之间执行必要的转换。

插件与您的系统隔离运行，这意味着出于安全原因，将不支持打印、读取文件或类似的操作。要作为插件运行，程序需要编译为 32 位共享 WebAssembly 库。许多编译器默认使用WASI ABI或者作为唯一的选项（例如 emscripten），它允许打印、读取文件等。此 ABI 不能直接与 Typst 一起使用。您需要编译到不同的目标或存根所有函数。

插件和包

插件以包的形式分发。包只需包含 WebAssembly 文件并加载即可使用插件。由于基于字节的插件接口相当低级，因此插件通常通过包装函数公开，这些函数也位于同一包中。

纯度

插件函数必须是纯的：给定相同的参数，它们必须始终返回相同的值。这样做的原因是 Typst 函数必须是纯函数（这对于语言设计来说非常重要），并且由于 Typst 函数可以调用插件函数，因此继承了这一要求。特别是，如果使用相同的参数调用插件函数两次，Typst 可能会缓存结果并仅调用您的函数一次。

例子

```
#let myplugin = plugin("hello.wasm")
#let concat(a, b) = str(
  myplugin.concatenate(
    bytes(a),
    bytes(b),
  )
)

#concat("hello", "world")
```

hello*world

协议

要用作插件，WebAssembly 模块必须符合以下协议：

出口

插件模块可以导出函数以使它们可以从 Typst 调用。为了符合协议，导出的函数应该：

- 采用n32 位整数参数`a_1`, `a_2`, ... `a_n`（解释为长度，因此`usize`/`size_t`可能更可取），并返回一个 32 位整数。
- `buf`该函数应首先分配一个长度为 的缓冲区`a_1 + a_2 + ... + a_n`，然后调用 `wasm_minimal_protocol_write_args_to_buffer(buf.ptr)`.
- `a_1`缓冲区的第一个字节现在构成第一个参数，接下来`a_2`的字节构成第二个参数，依此类推。
- 该函数现在可以使用参数完成其工作并生成输出缓冲区。在返回之前，它应该调用 `wasm_minimal_protocol_send_result_to_host`将其结果发送回主机。
- 为了表示成功，该函数应该返回`0`。
- 要发出错误信号，该函数应返回`1`。然后写入的缓冲区将被解释为 UTF-8 编码的错误消息。

进口

插件模块需要导入运行时提供的两个函数。 （类型和函数使用 WAT 语法描述。）

- `(import "typst_env" "wasm_minimal_protocol_write_args_to_buffer" (func (param i32)))`

  将当前函数的参数写入插件分配的缓冲区。当调用插件函数时，它接收输入缓冲区的长度作为参数。然后它应该分配一个缓冲区，其容量至少是这些长度的总和。然后，它应该使用 `aptr`来调用此函数，以将参数依次填充到缓冲区中。

- `(import "typst_env" "wasm_minimal_protocol_send_result_to_host" (func (param i32 i32)))`

  将当前函数的输出发送到主机 (Typst)。第一个参数应是指向缓冲区的指针 (`ptr`)，而第二个参数是该缓冲区的长度 (`len`)。`ptr`该函数返回后，可以立即释放指向的内存。如果该消息应被解释为错误消息，则应将其编码为 UTF-8。

资源

如需更多资源，请查看wasm-minimal-protocol 存储库。它包含了：

- 示例插件实现的列表以及这些示例的测试运行程序
- 帮助您用 Rust 编写插件的包装器（Zig 包装器正在开发中）
- WASI 的存根

构造函数

从 WebAssembly 文件创建一个新插件。

```
插入（斯特）->插入
```

path 斯特 *必需的* *位置性*

WebAssembly 文件的路径。