

WebAssembly预研：

1. WebAssembly.Module

`WebAssembly.Module` 对象包含已经由浏览器编译的无状态 WebAssembly 代码,可以高效地与 Workers 共享、缓存在 IndexedDB 中,和多次实例化。`WebAssembly.Module()` 构造函数可以用来同步编译给定的 WebAssembly 二进制代码。不过,获取 `Module` 对象的主要方法是通过异步编译函数 `WebAssembly.compile()`和`WebAssembly.compileStreaming()`：

- `WebAssembly.compile()`：编译一个二进制wasm代码到一个`WebAssembly.Module`对象。
- `WebAssembly.compileStreaming()`：从一个流式源中直接编译一个`WebAssembly.Module`对象。

一个二进制文件可以生成多个Module对象：

```
var module1, module2;
var importObj1 = {
  env: {
    memory: new WebAssembly.Memory({initial: 1, maximum: 10}),
  }
}
fetch('add.wasm').then(response =>
  response.arrayBuffer()
).then(bytes => {
  WebAssembly.instantiate(bytes,importObj1).then(
    res=>{
      module1 = res.module;
      console.log("module1 from instantiate:",module1);
    }
  )
  return WebAssembly.compile(bytes)
}
).then(module => {
  module2 = module;
  console.log("module2 from compile:", module2)
});
```

获得的Module对象只是记录了二进制文件有哪些导入导出属性,并不记录那些属性本身,因此还无法直接使用,需要使用`WebAssembly.instantiate()`或`WebAssembly.instantiateStreaming()`获取`WebAssembly.Instance`对象才能通过Instance对象使用二进制代码中的方法和数据。

```
console.log("module1===module2? ", module1 === module2, module1,
module2)
```

```
module1===module2? false ▼Module {} WebAssembly.Module
  ▶[[Prototype]]: WebAssembly.Module
  ▶[[Exports]]: Array(8)
    ▶0: {name: 'memory', kind: 'memory'}
    ▶1: {name: 'add', kind: 'function'}
    ▶2: {name: '___indirect_function_table', kind: 'table'}
    ▶3: {name: '___start', kind: 'function'}
    ▶4: {name: '___errno_location', kind: 'function'}
    ▶5: {name: 'stackSave', kind: 'function'}
    ▶6: {name: 'stackRestore', kind: 'function'}
    ▶7: {name: 'stackAlloc', kind: 'function'}
    length: 8
  ▶[[Prototype]]: Array(0)
  ▶[[Imports]]: Array(4)
    ▶0: {module: 'wasi_snapshot_preview1', name: 'args_sizes_get', kind: 'function'}
    ▶1: {module: 'wasi_snapshot_preview1', name: 'args_get', kind: 'function'}
    ▶2: {module: 'env', name: 'main', kind: 'function'}
    ▶3: {module: 'wasi_snapshot_preview1', name: 'proc_exit', kind: 'function'}
    length: 4
  ▶[[Prototype]]: Array(0)

▼Module {} WebAssembly.Module
  ▶[[Prototype]]: WebAssembly.Module
  ▶[[Exports]]: Array(8)
    ▶0: {name: 'memory', kind: 'memory'}
    ▶1: {name: 'add', kind: 'function'}
    ▶2: {name: '___indirect_function_table', kind: 'table'}
    ▶3: {name: '___start', kind: 'function'}
    ▶4: {name: '___errno_location', kind: 'function'}
    ▶5: {name: 'stackSave', kind: 'function'}
    ▶6: {name: 'stackRestore', kind: 'function'}
    ▶7: {name: 'stackAlloc', kind: 'function'}
    length: 8
  ▶[[Prototype]]: Array(0)
  ▶[[Imports]]: Array(4)
    ▶0: {module: 'wasi_snapshot_preview1', name: 'args_sizes_get', kind: 'function'}
    ▶1: {module: 'wasi_snapshot_preview1', name: 'args_get', kind: 'function'}
    ▶2: {module: 'env', name: 'main', kind: 'function'}
    ▶3: {module: 'wasi_snapshot_preview1', name: 'proc_exit', kind: 'function'}
    length: 4
  ▶[[Prototype]]: Array(0)
```

2.WebAssembly.Instance

`WebAssembly.Instance` 对象本身是有状态的，是 `WebAssembly.Module` 的一个可执行实例。`Instance` 包含所有的 `WebAssembly` 导出函数，允许从JavaScript 调用 `WebAssembly` 代码。一个 `WebAssembly.Module`对象可以生成多个`WebAssembly.Instance`对象：

```
var instance1, instance2;
var module1, module2;
var importObj1 = {
  env: {
    memory: new WebAssembly.Memory({
      initial: 1,
      maximum: 10
    }),
    function: "_add",
    a: ()=>{}
  }
}
var importObj2 = {
  env: {
    memory: new WebAssembly.Memory({
      initial: 128,
      maximum: 1024
    }),
  }
}
fetch('add.wasm').then(response =>
  response.arrayBuffer())
.then(bytes => WebAssembly.compile(bytes))
.then(module => {
  WebAssembly.instantiate(module, importObj1)
    .then(function (instance) {
      instance1 = instance
      console.log("instance1 :", instance1)
    });

  WebAssembly.instantiate(module, importObj1)
    .then(function (instance) {
      instance2 = instance
      console.log("instance2 :", instance2)
    });
});
```

instance1 === instance2? false ▼ Instance {exports: {...}} ⓘ

```
▼ exports:
  ▶ add: f $add()
  ▶ memory: Memory(256) ⓘ
  ▶ stackAlloc: f $stackAlloc()
  ▶ stackRestore: f $stackRestore()
  ▶ stackSave: f $stackSave()
  ▶ __errno_location: f $__errno_location()
  ▶ __indirect_function_table: Table {length: 2}
  ▶ _start: f $_start()
  ▶ [[Prototype]]: WebAssembly.Instance
  ▶ [[Module]]: Module
  ▶ [[Functions]]: Functions
  ▶ [[Globals]]: Globals
  ▶ [[Memories]]: Memories
  ▶ [[Tables]]: Tables
```

▼ Instance {exports: {...}} ⓘ

```
▼ exports:
  ▶ add: f $add()
  ▶ memory: Memory(256) ⓘ
  ▶ stackAlloc: f $stackAlloc()
  ▶ stackRestore: f $stackRestore()
  ▶ stackSave: f $stackSave()
  ▶ __errno_location: f $__errno_location()
  ▶ __indirect_function_table: Table {length: 2}
  ▶ _start: f $_start()
  ▶ [[Prototype]]: WebAssembly.Instance
  ▶ [[Module]]: Module
  ▶ [[Functions]]: Functions
  ▶ [[Globals]]: Globals
  ▶ [[Memories]]: Memories
  ▶ [[Tables]]: Tables
```

二者是不同的对象，各自有各自的导出属性以及线性内存区域，改变其中一个Instance的线性内存区域不会影响另一个Instance：

```

console.log(
  "instance1 === instance2? ",
  instance1 === instance2, instance1,
  instance2);
let arr1 = new Uint8Array(instance1.exports.memory.buffer, 0, 800000)
arr1.fill(2)

```

```

instance1 === instance2? false ▼ Instance {exports: {...}} ⓘ
  ▼ exports:
    ▶ add: f $add()
    ▼ memory: Memory(256) ⓘ
      ▼ buffer: ArrayBuffer(16777216) ⓘ
        byteLength: 16777216
        ▶ [[Prototype]]: ArrayBuffer
        ▼ [[Int8Array]]: Int8Array(16777216)
          ▼ [0 ... 999999]
            ▼ [0 ... 99]
              0: 2
              1: 2
              2: 2
              3: 2
              4: 2
              5: 2
              6: 2
              7: 2
              8: 2
              9: 2
              10: 2
          ▼ [0 ... 999999]
            ▼ [0 ... 99]
              0: 0
              1: 0
              2: 0
              3: 0
              4: 0
              5: 0
              6: 0
              7: 0
              8: 0
              9: 0
              10: 0

```

3.在不同Module产生的Instance之间共享线性内存

- 动态链接