# WebAssembly 实践

京程一灯 · 做有深度的精品课

http://www.yidengxuetang.com

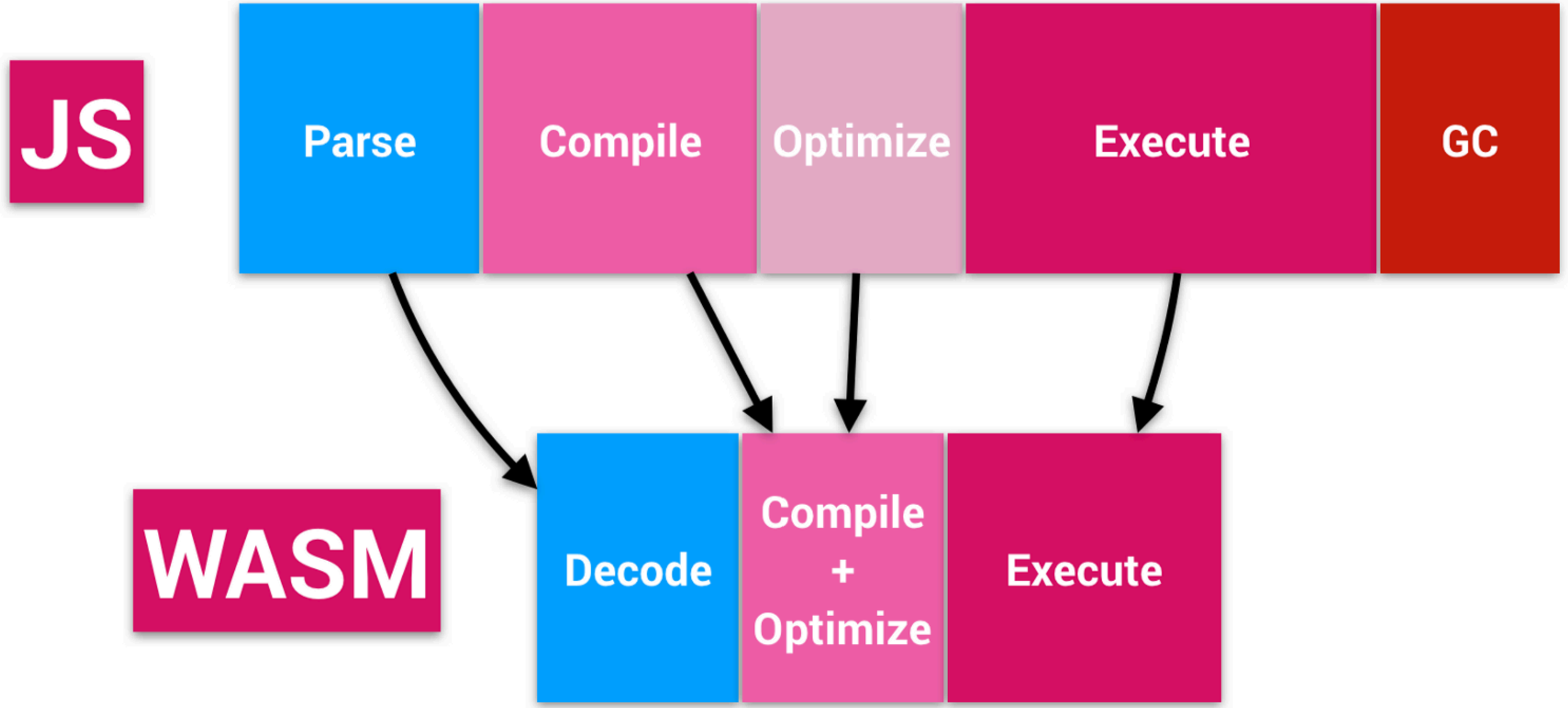
# 主要内容

- WASM的特点
- 一窥底层原理
- 性能表现
- WASM 的开发工具
- JavaScript API
- C 语言编译成 WebAssembly
- C 语言实现一个 Hello World

# WASM 的特点

- 高效
  - WebAssembly 有一套完整的语义，实际上 wasm 是体积小且加载快的二进制格式， 其目标就是充分发挥硬件能力以达到原生执行效率
- 安全
  - WebAssembly 运行在一个沙箱化的执行环境中，甚至可以在现有的 JavaScript 虚拟机中实现。在web环境中，WebAssembly将会严格遵守同源策略以及浏览器安全策略。
- 开放
  - WebAssembly 设计了一个非常规整的文本格式用来、调试、测试、实验、优化、学习、教学或者编写程序。可以以这种文本格式在web页面上查看wasm模块的源码。
- 标准
  - WebAssembly 在 web 中被设计成无版本、特性可测试、向后兼容的。WebAssembly 可以被 JavaScript 调用，进入 JavaScript 上下文，也可以像 Web API 一样调用浏览器的功能。当然，WebAssembly 不仅可以运行在浏览器上，也可以运行在非web环境下。

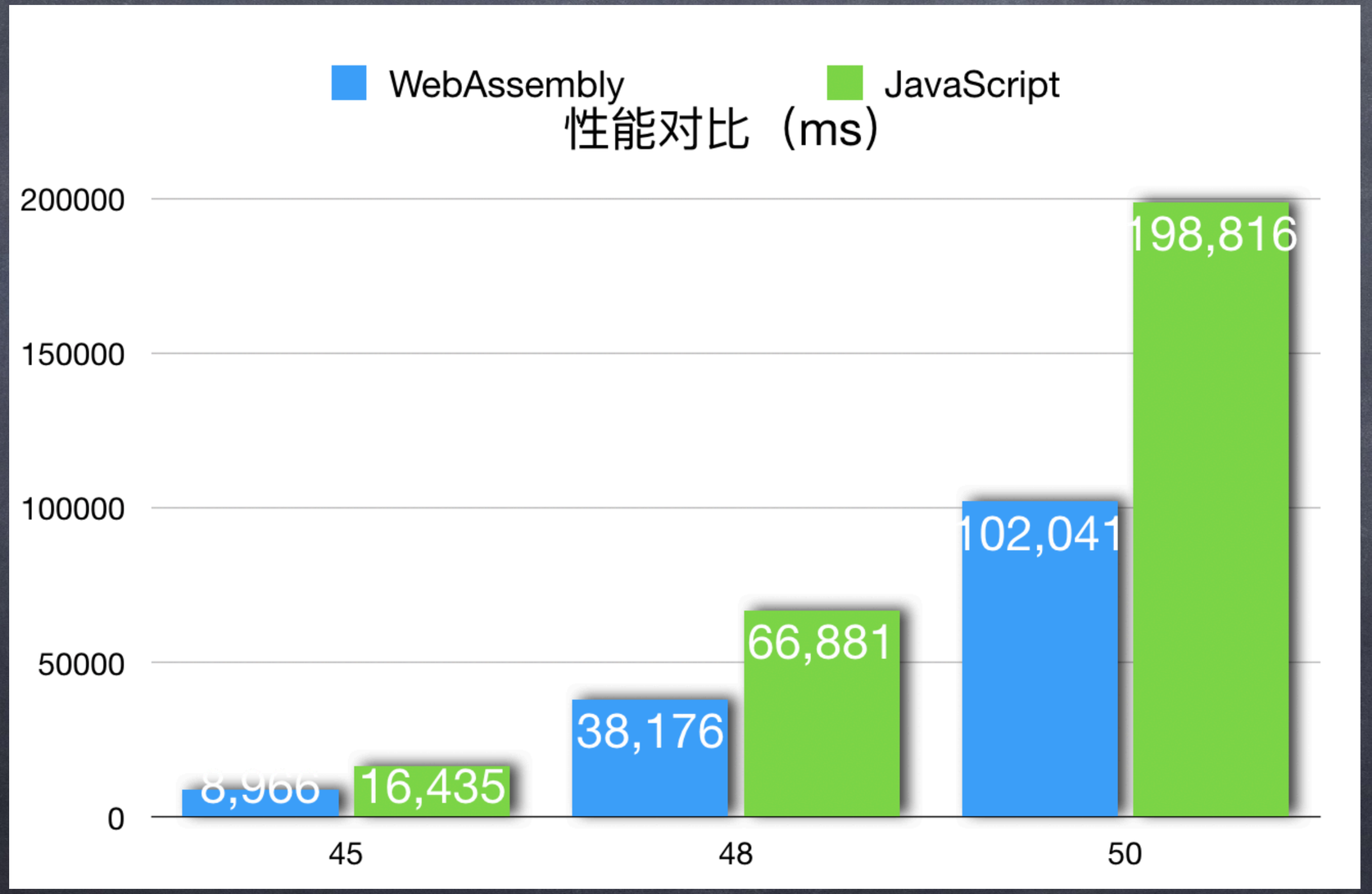

C++, C or Rust
WASM
JS
CSS
HTML


JS
Parse
Compile
Optimize
Execute
GC
WASM
Decode
Compile + Optimize
Execute

# WASM 的工具

- AssemblyScript：支持直接将 TypeScript 编译成 WebAssembly。这对于前端来说，入门的门槛很低的。
- Emscripten：可以说是 WebAssembly 的灵魂工具。将其他的高级语言，编译成 WebAssembly。
- WABT：将 WebAssembly在字节码和文本格式相互转换的一个工具，方便开发者去理解这个 wasm 到底是在做什么事。

WebAssembly版本和原生JavaScript版本的递归无优化的Fibonacci函数，下图是这两个函数在值是45、48、50的时候的性能对比。



# JavaScript API

方法

- 1.WebAssembly.compile()
- 2.WebAssembly.instantiate()
- 3.WebAssembly.validate()

类

- 1.WebAssembly.Module
- 2.WebAssembly.Instance
- 3.WebAssembly.Memory
- 4.WebAssembly.Table
- 5.WebAssembly.CompileError
- 6.WebAssembly.LinkError
- 7.WebAssembly.RuntimeError

WebAssembly.compile()

```
Promise<WebAssembly.Module> WebAssembly.compile(bufferSource);
```

WebAssembly.Module

```
var myModule = new WebAssembly.Module(bufferSource);
```

## WebAssembly.instantiate()

Promise<ResultObject>

```
WebAssembly.instantiate(bufferSource, importObj);
```

ResultObject:{module, instance}

Promise<WebAssembly.Instance>

```
WebAssembly.instantiate(module, importObject);
```

## WebAssembly.Instance

```
var myInstance = new WebAssembly.Instance(module, importObject);
```

# WebAssembly.validate()

```
WebAssembly.validate(bufferSource);
```

Result : true/false

# WebAssembly.Memory

可用于 JavaScript 和 WebAssembly 的数据共享

可以使用JS创建，传递给 WASM

可以在WASM里创建，使用JS获取

# WebAssembly.Table

可以用来在JS对象上存放WASM函数的引用

未来可能有更多的用途

WebAssembly.CompileError

WebAssembly.LinkError

WebAssembly.RuntimeError

遇到的时候解决了就知道是什么原因导致的啦！

# 开始使用 JS API

未来会支持这样引入

```
import module from 'test.wasm';
```

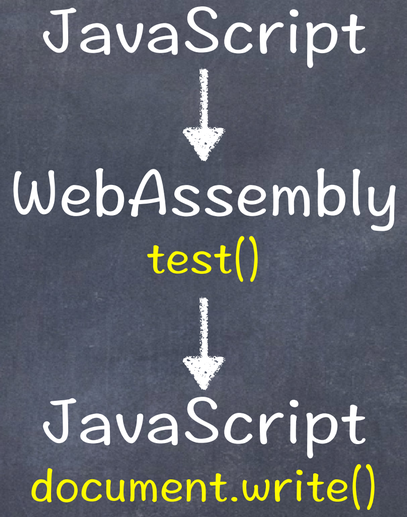
目前还不支持

可以使用 webpack 的 wasm-loader 编译

```
const importObject = {
  func: {
    log: (num)=> document.write(num)
  }
}
fetch('test.wasm')
.then(response=>response.arrayBuffer())
.then(types=>WebAssembly.instantiate(bytes, importObject))
.then(({instance}) => window.test = instance.exports.test)
```

目前我们这么引入一个WASM模块

# 这个过程是这样的




那 WebAssembly 是如何生成的呢？

使用 文本格式 编写 WebAssembly

文本格式 基于S-表达式

可以用于调试、学习、手写WebAssembly

test.wat 文件

```
(module
  (func
    $log
  )
  (func)(export "test")
    i32.const 1234567890
    call $log
  )
)
```

```
const importObject = {

  func: {

    log: (num)=> document.write(num)

  }

}
```

# 使用WABT来编译S-表达式

The WebAssembly Binary Toolkit

wast2wasm
wasm2wast
...
https://github.com/WebAssembly/wabt

```
wast2wasm test.wat -o test.wasm
```

我们得到一个 WASM 文件

一个二进制文件

直到这里

你已经知道WA和JS如何相互调用了

# C语言编译成 WebAssembly

# 推荐方式

## Emscripten

相对复杂，需要配置

https://emscripten.org/docs/getting_started/downloads.html

# 在线版

https://wasdk.github.io/WasmFiddle/

相互调用的🌰

https://wasdk.github.io/WasmFiddle/?e2wet

# Hello World

https://wasdk.github.io/WasmFiddle/?cevx1

WebAssembly 中文网

http://webassembly.org.cn