D2 前端技术论坛
D2 FRONTEND TECHNOLOGY FORUM
精心

第十六届D2前端技术论坛

V8 JS AOT化的探索与实践

喻世江

D2 前端技术论坛
D2 FRONTEND TECHNOLOGY FORUM
精心

Contents
目录

01

背景

背景

U4内核V8引擎

快

- ✓ Disk Code Cache
- ✓ UC LLVM Compiler

强

- ✓ JS 卡死检测
- ✓ OOM 定位信息
- ✓ JS API 扩展
- ✓ JSI

稳

- ✓ 疑难崩溃攻克
- ✓ 安全漏洞修复

	JavaScript		Native
跨平台	☺		☹
动态化	☺		☹
性能	☹		☺
	➢ 运行时在线编译 ➢ 每次重新编译		➢ 打包时 PC 离线编译 ➢ 直接运行

背景

JS AOT

AOT：Ahead of Time，提前编译。

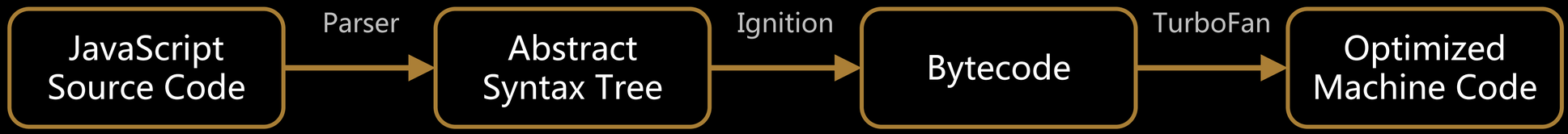
目标

让 JavaScript 具备动态化特性的同时，运行性能也可能与 Native 对标，尤其是首次启动。

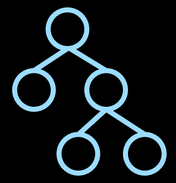
02

技术选型

V8 运行 JavaScript 的流水线



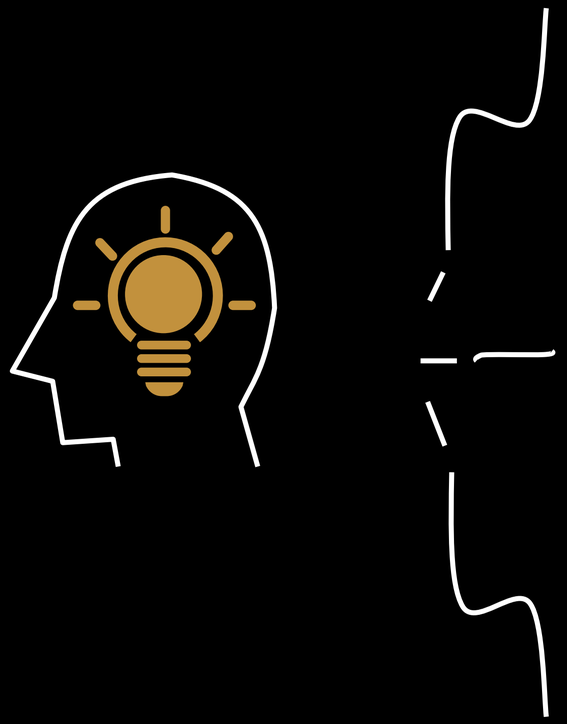

```
function test(o) {
  return 3 + o.x;
}
```



```
StackCheck
LdaSmi [3]
Star r0
LdaNamedProperty a0, [0], [1]
Add r0, [0]
Return
```

```
REX.W leaq rbx,[rip+0xffffff9]
REX.W cmpq rbx,rcx
jz 0x1efffc744 <+0x24>
REX.W movq rdx,0x3e00000
REX.W movq r10,0x7f2be00
call r10
int3l
REX.W movq rbx,[rcx-0x20]
testb [rbx+0xf],0x1
...
```

技术选型



本地代码

（Machine Code，汇编）

全字节码缓存

(Full Code Cache)

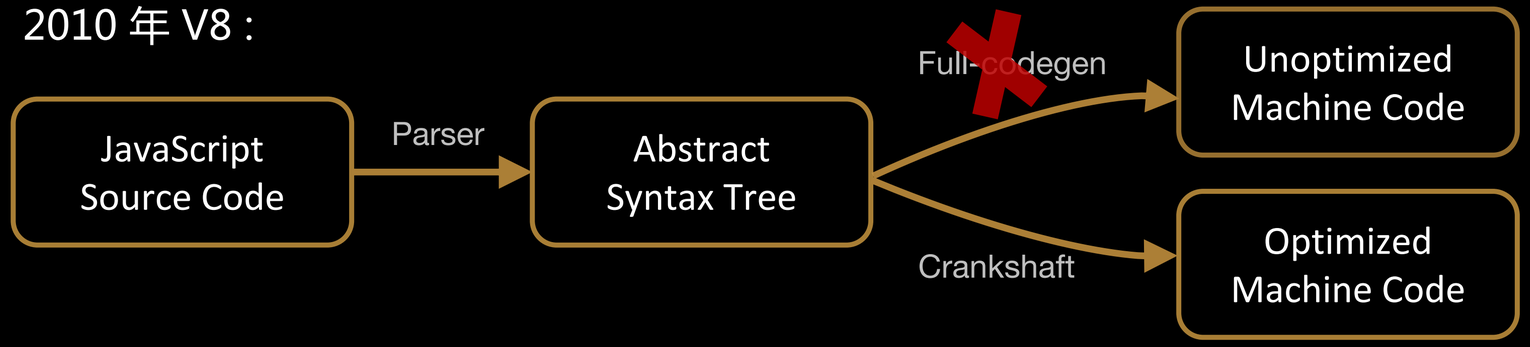
部分字节码缓存

(Code Cache)

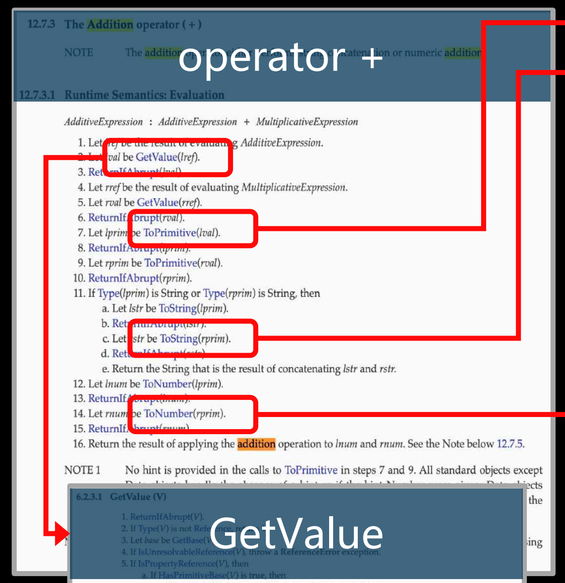
技术选型

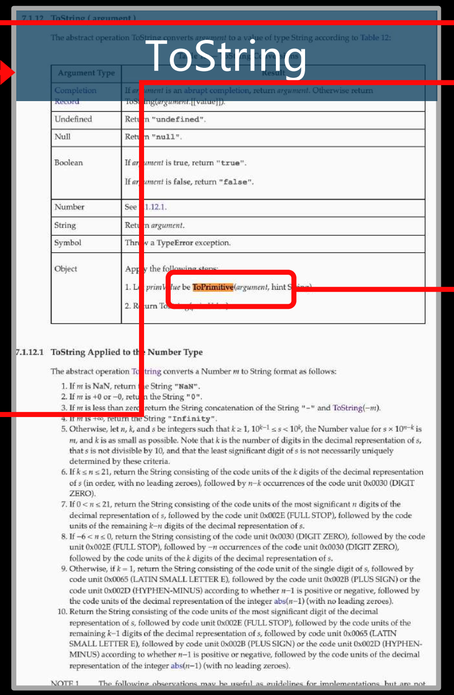
调研一：使用本地代码（ Machine Code，汇编 ）

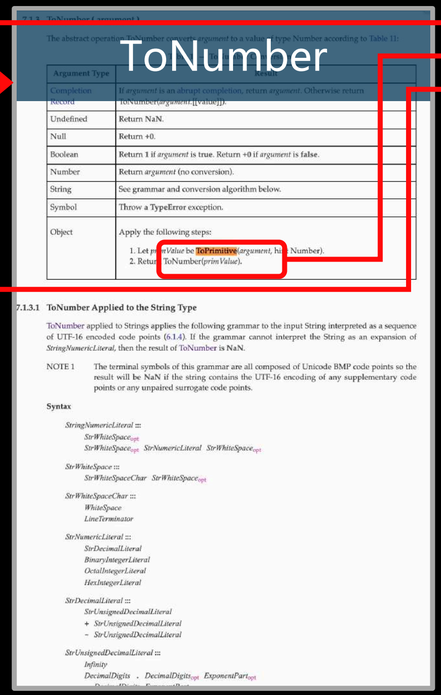
2010 年 V8 :

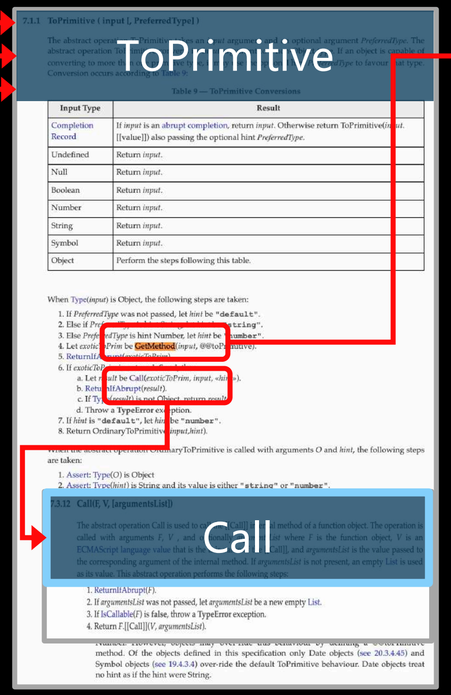


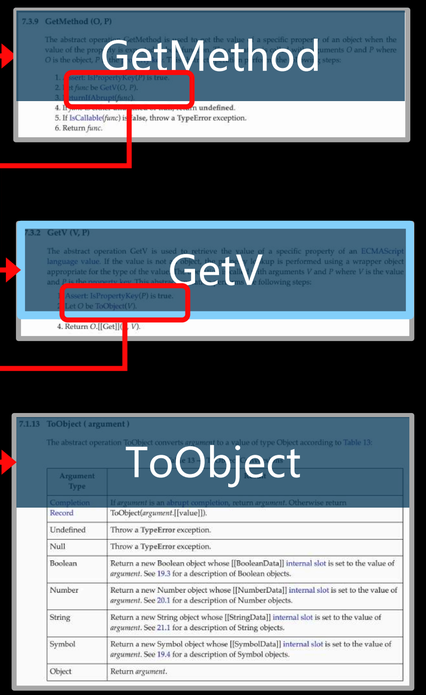
- ☹ 编译慢
- ☹ 体积大
- ☹ 内存高
- ☹ 性能差
- ☹ CPU 架构不通用

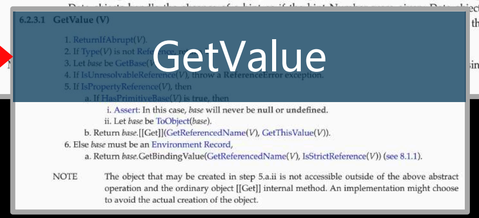












技术选型

调研二：缓存全部字节码（Full Code Cache）

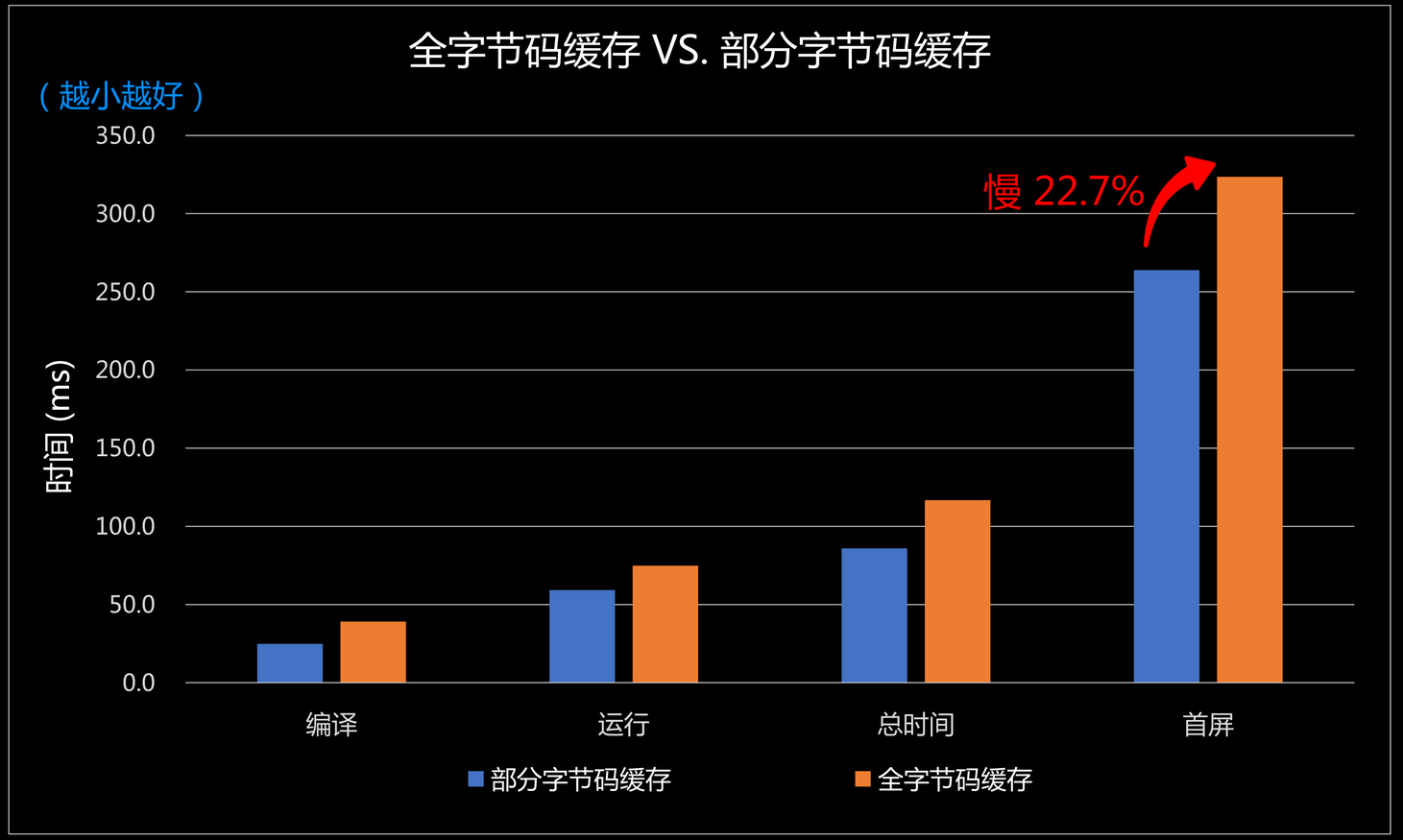
- JS 函数运行覆盖度低（43%）

$$= \frac{\text{运行函数个数}}{\text{总函数个数}}$$

- 代码膨胀严重（2.6 x）

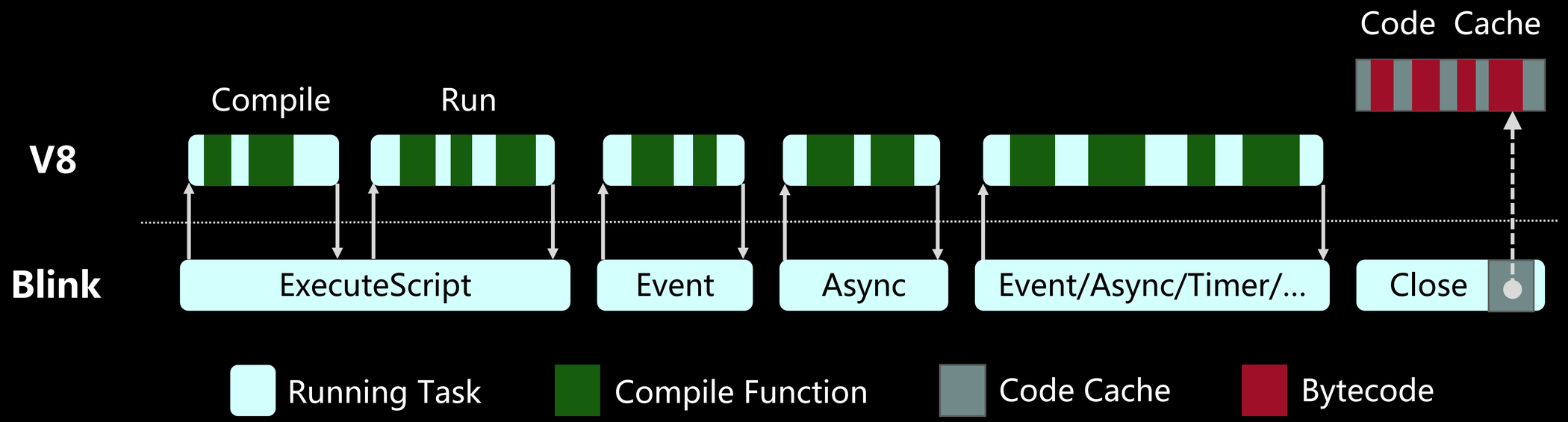
$$= \frac{\text{全字节码大小}}{\text{JS源码大小}}$$

- 加载 & 反序列化消耗大



技术选型

调研三：缓存部分字节码（Code Cache）



- ☹ V8 (U4 内核) 版本的碎片化
- ☹ CPU 架构通用性
- ☹ 机型通用性

面临的主要难题

有效性

兼容性

03

方案设计

保证 AOT 的

有效性

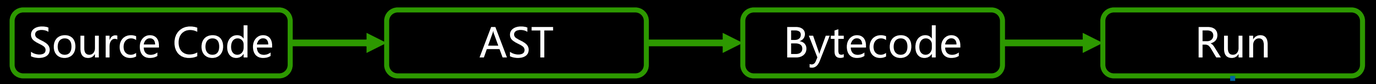
方案设计

保证有效性

策略一：追求极致的性能 —— PGO（Profile-guided optimization）

仅为需要被执行的函数生成代码。

函数信息收集
（移动端预发布）



上报

函数信息处理
（服务端）


```
{
  "s": 12, // 函数起始
  "e": 26  // 函数终止
}
```

```
{
  "name": "foo", // 函数名
  "start": 12, // 函数起始
  "end": 26,  // 函数终止
  "count": 1, // 运行次数
}
```

AOT 生成

方案设计

保证有效性

策略二：追求便捷的使用 —— 先验规则

预测需要被执行的函数。

```
function foo() { /* ... */ }
function bar() {
    function inner() {
        return (x, y) => {
            return x + y;
        }
    }
    return inner();
}
var Test = {
    "a": function() { /* ... */ },
    "b": function() {
        return () => {
            return "hello";
        }
    },
}
```

特征

- 嵌套越深，使用率越低
- JS 越小，运行覆盖度越高

策略

- 小型 JS：不生成
- 中型 JS：全字节码缓存（覆盖度 >80%）
- 大型 JS：Top 3 层
- 运行后增量更新

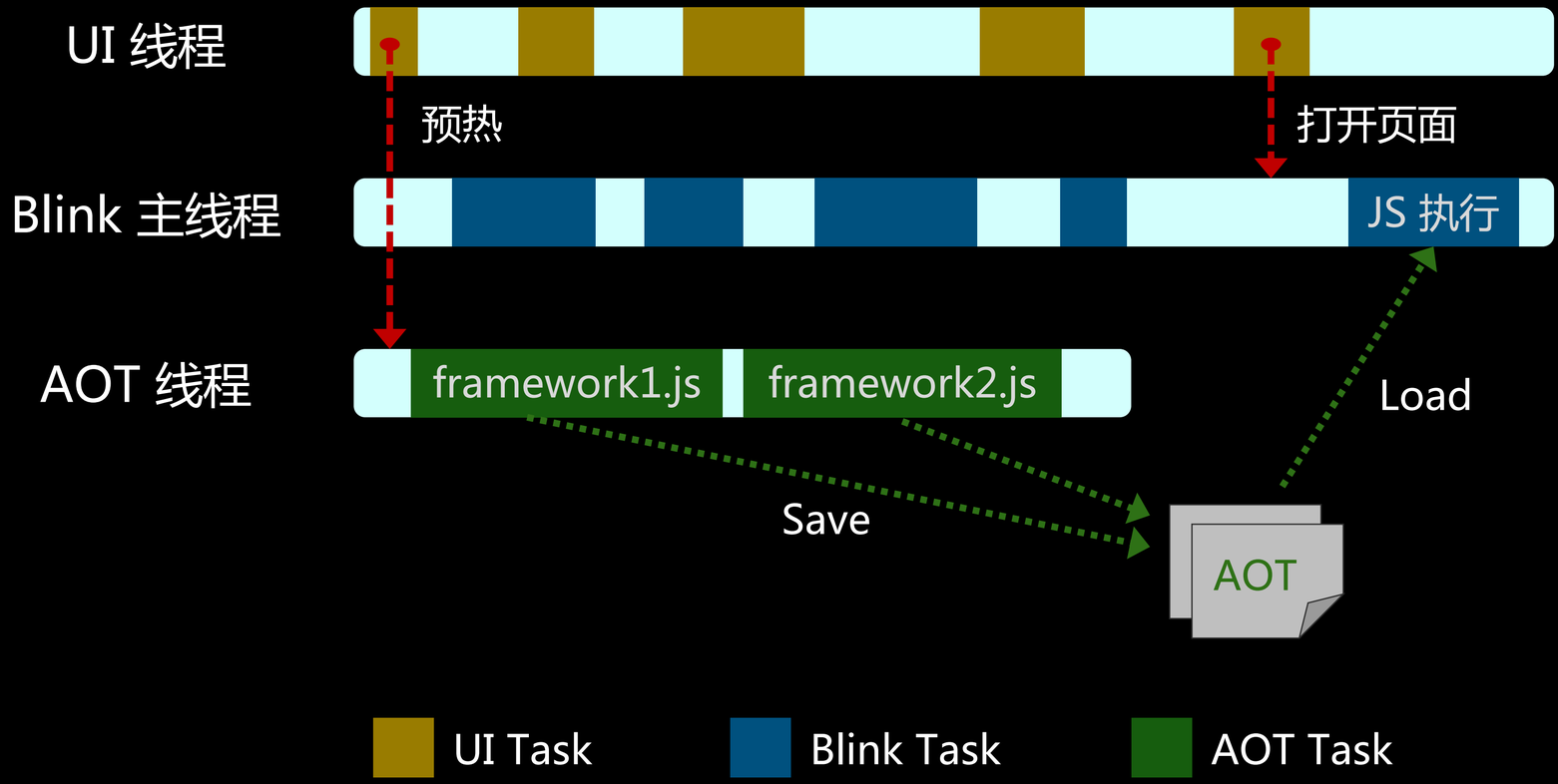
保证 AOT 的

兼容性

方案设计

保证兼容性

策略一：在线生成（空闲时预热）



生成时机：

- APP 空闲时
- 后台线程

影响或不足：

- 资源浪费（磁盘 & CPU）

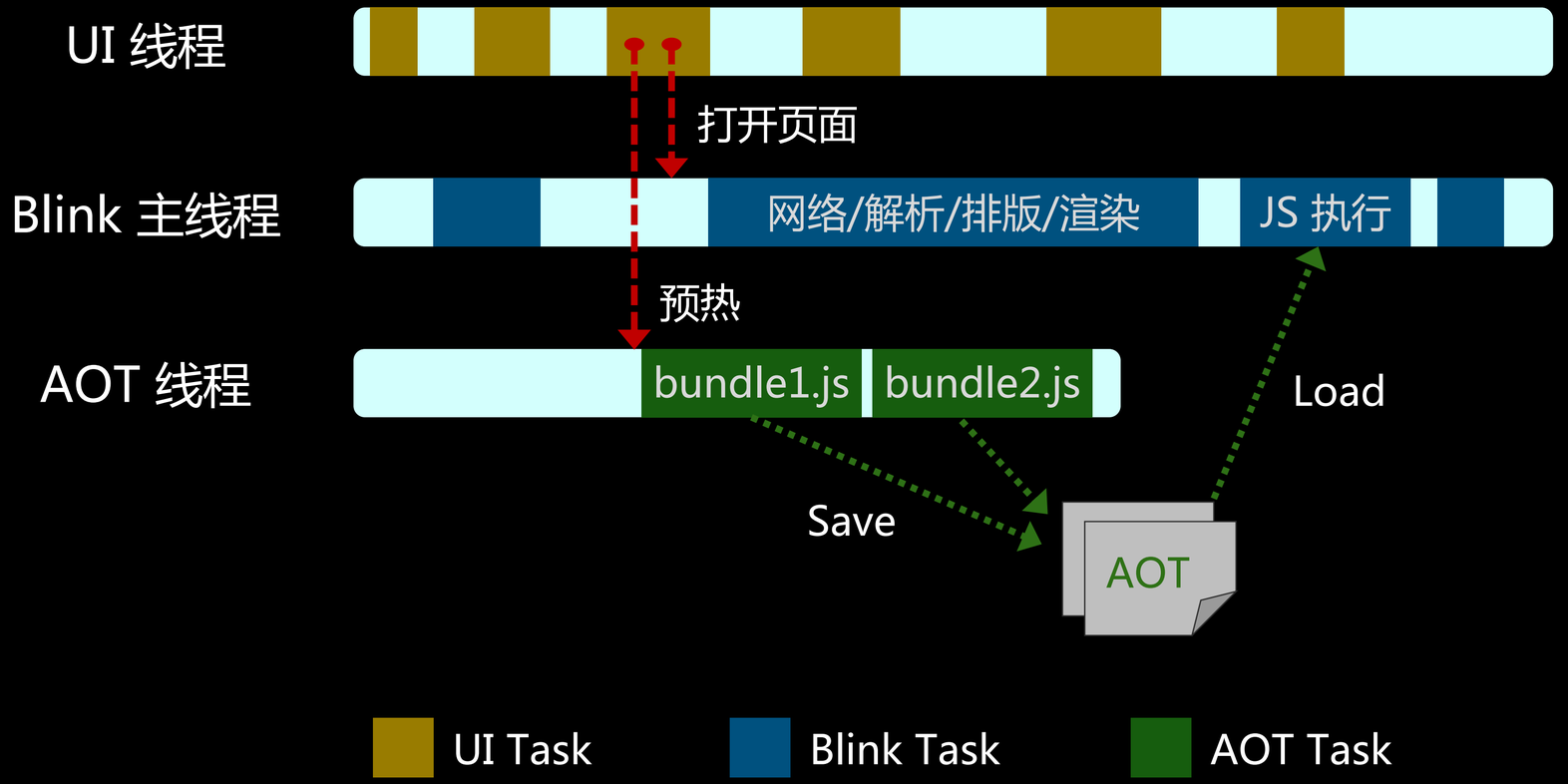
适用场景：

- 框架 JS，不经常变动

方案设计

保证兼容性

策略一：在线生成（访问时生成）



生成时机：

- 页面打开时
- 后台线程

影响或不足：

- 可能来不及

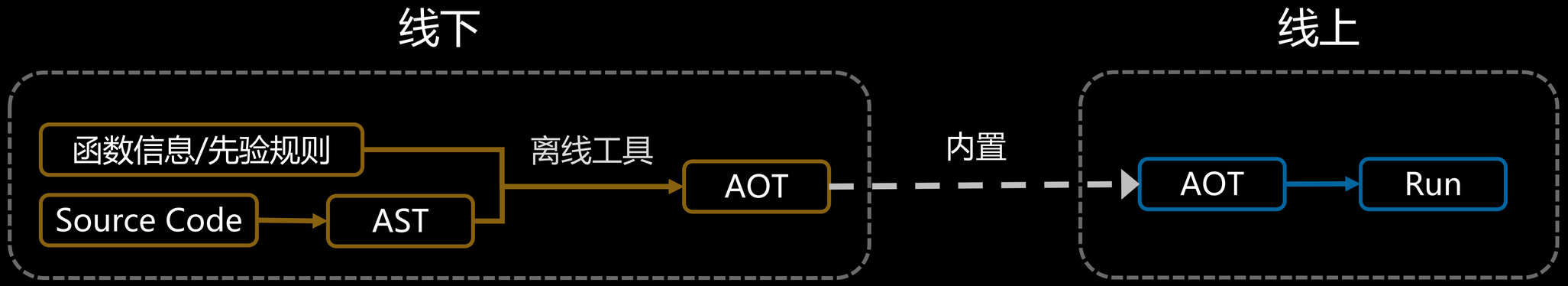
适用场景：

- 业务 JS，已经离线到本地

方案设计

保证兼容性

策略二：离线生成



适用场景：

- APP冷启动时执行（没有预热时机）
- 不经常变动
- 不需要动态更新

影响或不足：

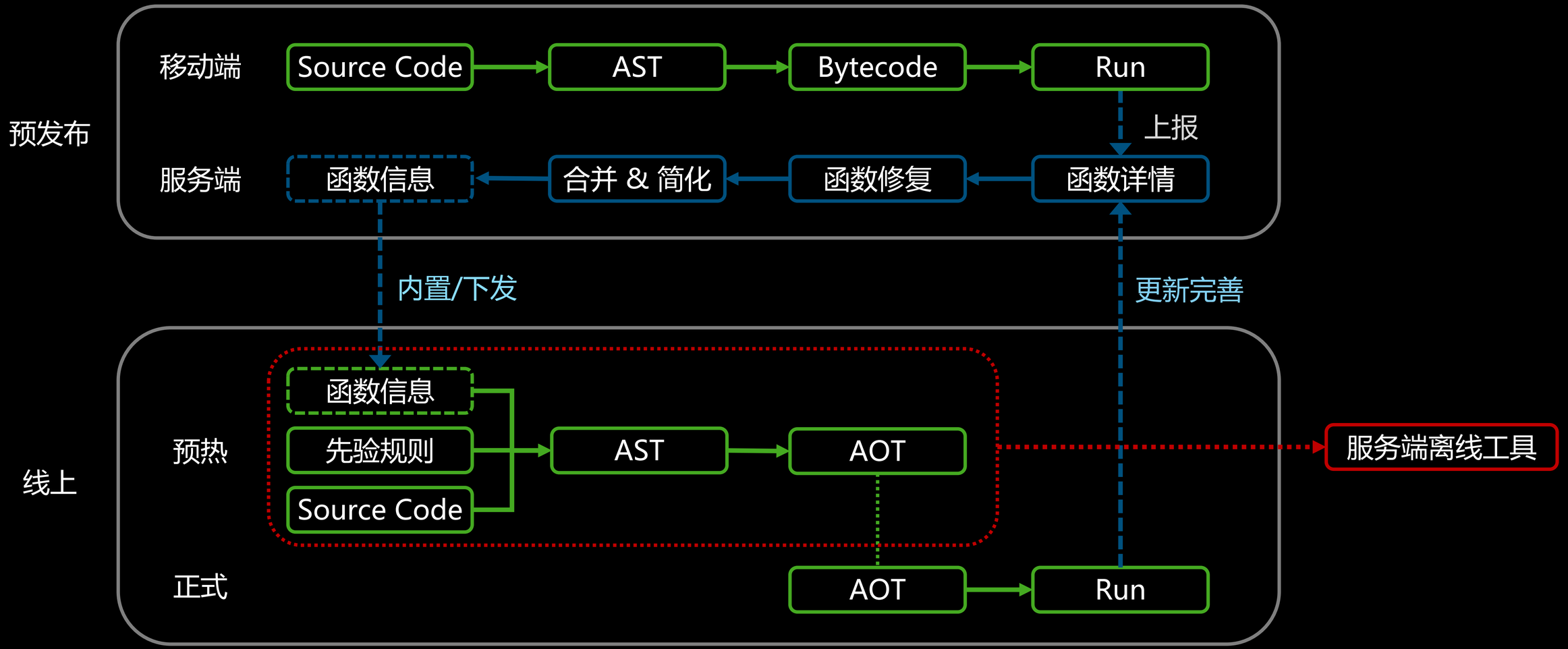
- 更新 U4 内核后，可能需同步更新 AOT

丢弃 JS 源码？

- Full Code Cache
- Function.prototype.toString()

方案设计

AOT 总览


预发布
移动端
Source Code
AST
Bytecode
Run
上报
服务端
函数信息
合并 & 简化
函数修复
函数详情
内置/下发
更新完善
线上
预热
函数信息
先验规则
Source Code
AST
AOT
服务端离线工具
正式
AOT
Run

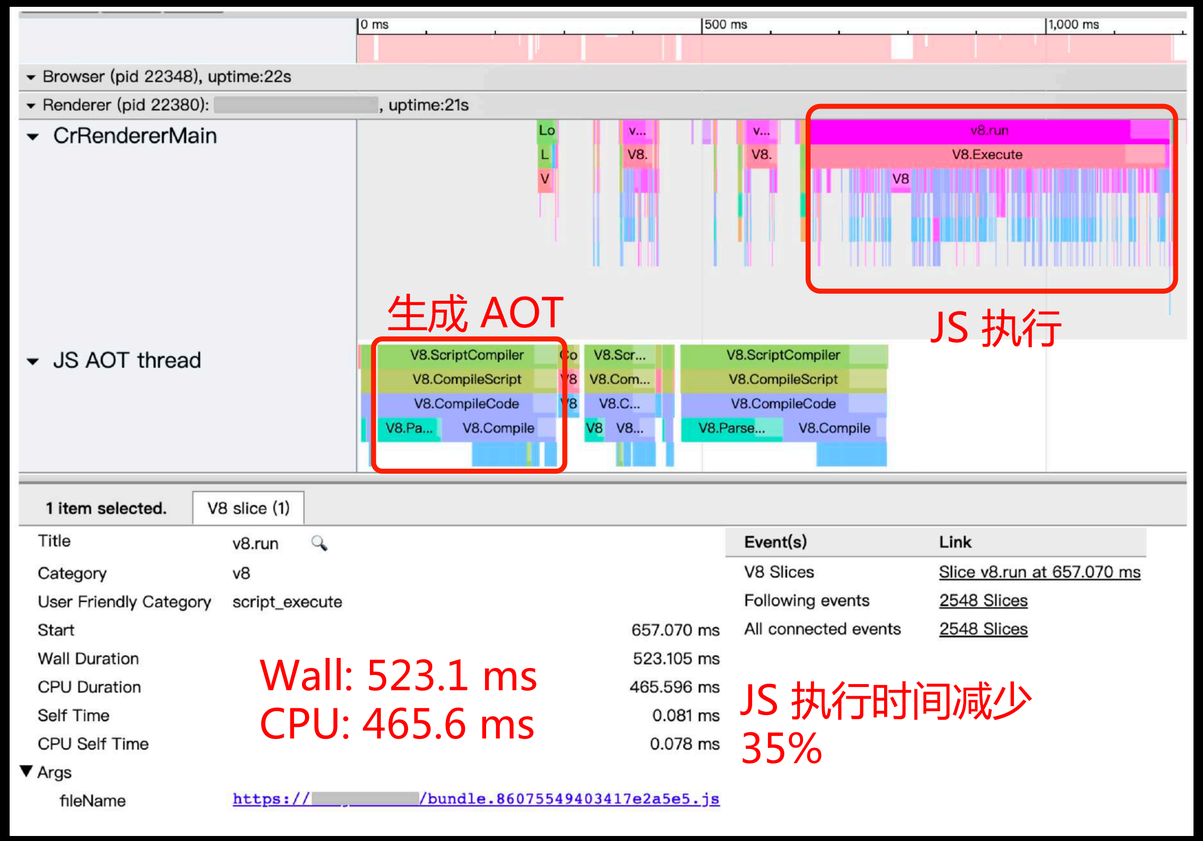
04

优化效果

优化效果

正常访问（无 AOT）

打开时预热生成 AOT（先验规则）

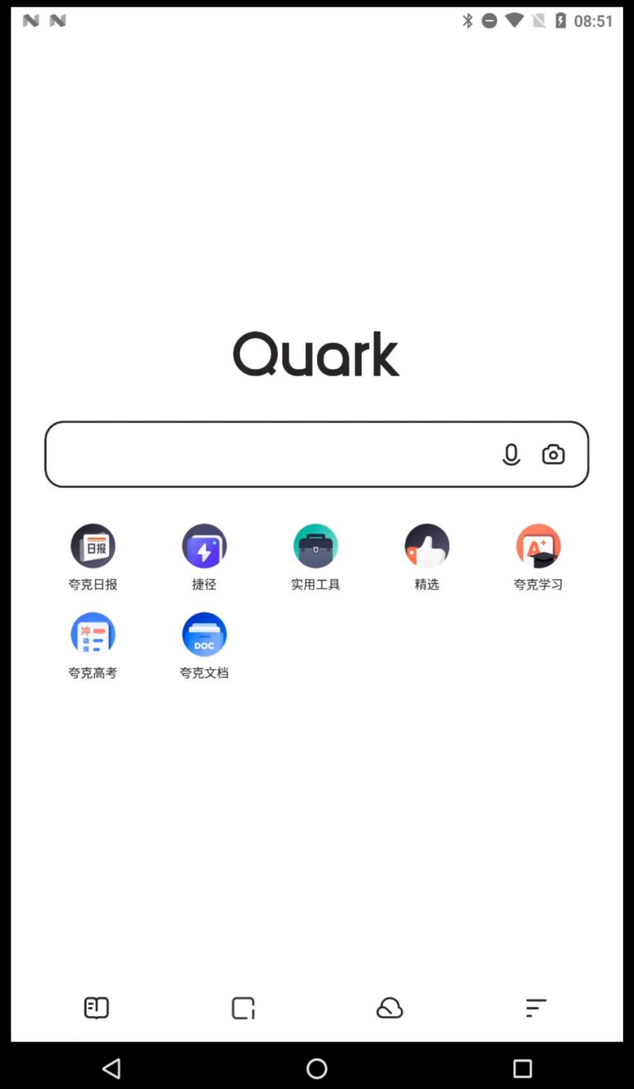


PGO	49.0% ↑
先验规则	33.9% ↑

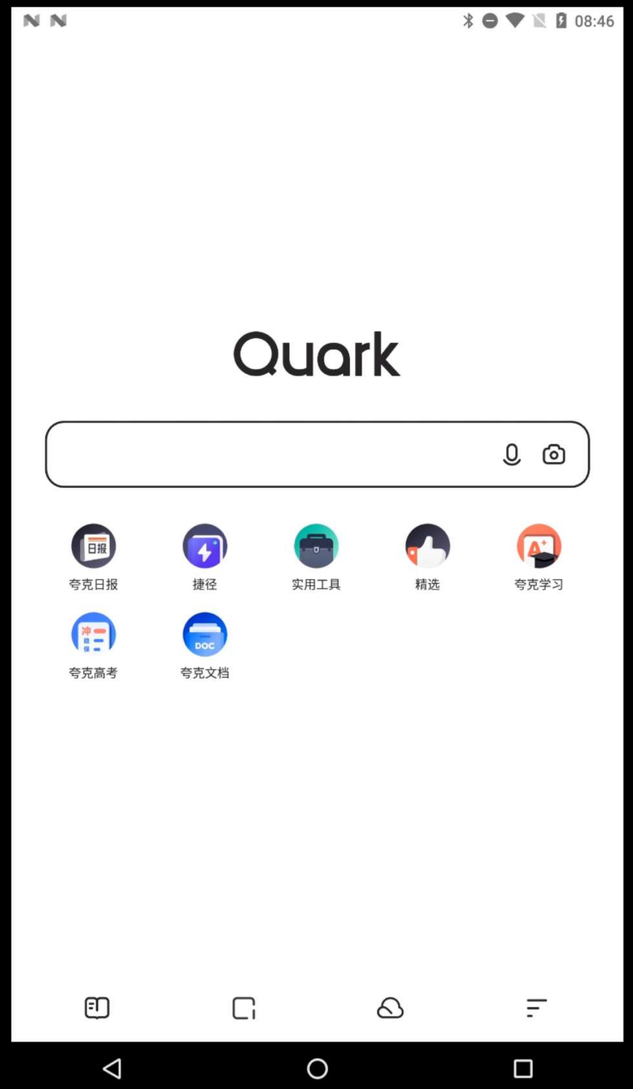
(TOP 30+ 站点平均)

夸克高考首屏性能提升 17.6%

无 AOT



使用 AOT

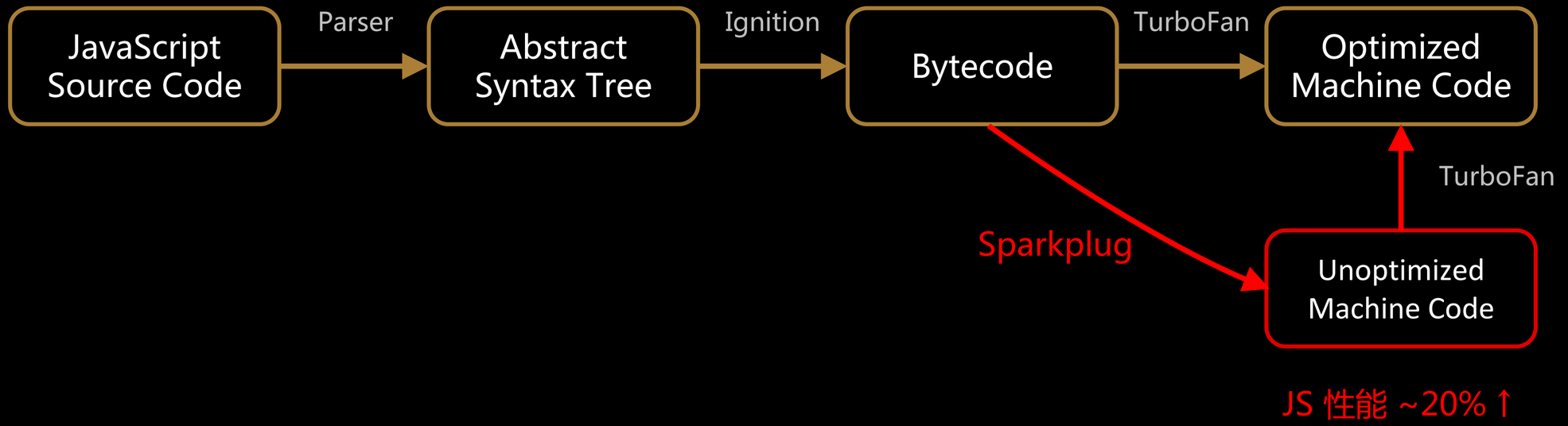


05

展 望

展望

U4 3.0 & 4.0 ⇨ U4 5.0



JS 性能 ~20% ↑

Thanks

“U4 内核技术”公众号