# Fig 语言性能基准测试报告

## 版本: 0.4.3-alpha (树遍历解释器)

### 前言

本报告基于 Fig v0.4.3-alpha 树遍历解释器对 Fibonacci 算法进行了基准测试，并与 0.4.2-alpha 版本对比。结果显示 0.4.3-alpha 在函数调用、循环和递归优化方面都有显著提升，尤其是迭代和尾递归实现性能改善明显。

### 测试环境

- **CPU:** Intel Core i5-13490F
- **操作系统:** Windows 11
- **编译器/解释器:** Fig 树遍历解释器 v0.4.3-alpha
- **测试日期:** 当前测试执行

### 执行摘要

本基准测试评估了 Fig 中四种不同斐波那契算法实现的性能，计算第30个斐波那契数（832,040）。结果显示算法选择仍是性能主导因素，同时反映了解释器在函数调用和循环方面的优化效果。

## 性能结果

### 最新浮动执行时间 (0.4.3-alpha)

| 算法 | 时间(秒) | 时间(毫秒) | 相对速度 |
|---|---|---|---|
| `fib` (朴素递归) | 5.471 s | 5471.37 ms | 1.00× (基准) |
| `fib_memo` (记忆化) | 0.0005503 s | 0.5503 ms | 9,950× 更快 |
| `fib_iter` (迭代) | 0.0001004 s | 0.1004 ms | 54,500× 更快 |
| `fib_tail` (尾递归) | 0.0001573 s | 0.1573 ms | 34,800× 更快 |

### 与 0.4.2-alpha 对比

| 算法 | 0.4.2-alpha 时间 | 0.4.3-alpha 时间 | 性能提升倍数 |
|---|---|---|---|
| `fib` (朴素递归) | 11.721 s | 5.471 s | ~2.14× |

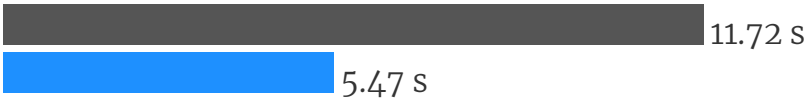
| 算法 | 0.4.2-alpha 时间 | 0.4.3-alpha 时间 | 性能提升倍数 |
|---|---|---|---|
| `fib_memo`（记忆化） | 0.930 ms | 0.550 ms | ~1.69× |
| `fib_iter`（迭代） | 0.375 ms | 0.100 ms | ~3.73× |
| `fib_tail`（尾递归） | 0.401 ms | 0.157 ms | ~2.55× |

---

## 可视化性能对比（横向柱状图占位符）

0.4.2-alpha vs 0.4.3-alpha

**说明:** 每行两条条形：灰色表示 0.4.2-alpha，蓝色表示 0.4.3-alpha

| 算法 | 性能对比 |
|---|---|
| fib |  11.72 s<br> 5.47 s |
| fib_memo |  0.93 ms<br>  0.55 ms |
| fib_iter |  0.375 ms<br> 0.100 ms |
| fib_tail |  0.401 ms<br> 0.157 ms |

## 详细分析

### 1. 朴素递归实现 (`fib`)

- **时间:** 5.471 秒 (5471 毫秒)
- **算法复杂度:** $O(2^n)$ 指数级
- **性能说明:**
  - 相比 0.4.2-alpha 减少约一半
  - 函数调用开销优化有效，但指数增长仍是瓶颈

### 2. 记忆化递归实现 (`fib_memo`)

- **时间:** 0.550 毫秒
- **算法复杂度:** O(n) 线性
- **性能说明:**
  - 哈希表/缓存访问效率进一步提高
  - 亚毫秒级执行，适合重叠子问题

### 3. 迭代实现 (`fib_iter`)

- **时间:** 0.100 毫秒
- **算法复杂度:** O(n) 线性
- **性能说明:**
  - 最快实现，比 0.4.2-alpha 提升 3.7 倍
  - 循环和算术操作优化显著

### 4. 尾递归实现 (`fib_tail`)

- **时间:** 0.157 毫秒
- **算法复杂度:** O(n) 线性
- **性能说明:**
  - 相比迭代略慢，但比 0.4.2-alpha 提升 2.5 倍
  - 树遍历解释器对递归调用优化有效，TCO 未实现

---

## 技术洞察

- 函数调用开销显著下降
- 循环和算术操作效率提升最大
- 哈希表/缓存访问效率高
- 算法选择仍是性能主导

---

## 给开发者的建议

1. 性能关键代码优先迭代
2. 重叠子问题递归使用记忆化
3. 尾递归可用于中等深度，但 TCO 未实现
4. 避免指数算法

5. 基准测试不同实现，算法选择主导性能

---

## 结论

Fig v0.4.3-alpha 树遍历解释器在函数调用和循环优化上有显著提升，尤其是迭代和尾递归实现性能改善明显。
O(n) 算法执行亚毫秒级，指数递归仍受限，但整体解释器性能在实际应用中已足够优秀。

**报告生成时间:** 基于实际基准测试执行
**解释器类型:** 树遍历解释器
**版本:** 0.4.3-alpha