Vue 3.0 进展分享

尤雨溪

VueConf 杭州, Nov. 2018

Vue 3.0 会带来些什么？

- 更快
- 更小
- 更易于维护
- 更好的多端渲染支持
- 新功能

更快

新的 Virtual DOM 实现
完全重构

初始渲染/更新提速达 100%

更多编译时的优化
以减少运行时的开销

Component fast path + Monomorphic calls + Children type detection

Template

```
<Comp></Comp>
<div>
  <span></span>
</div>
```

Compiler output

```
render() {
 const Comp = resolveComponent('Comp', this)
 return createFragment([
   createComponentVNode(Comp, null, null, 0 /* no children */),
   createElementVNode('div', null, [
     createElementVNode('span', null, null, 0 /* no children */)
   ], 2 /* single vnode child */)
 ], 8 /* multiple non-keyed children */)
}
```

- 跳过不需要的条件判断
- 生成更易于被 JS 引擎优化的代码

优化 Slots 生成

Template

```
<Comp>
  <div>{{ hello }}</div>
</Comp>
```

Compiler output

```
render() {
 return h(Comp, null, {
   default: () => [h('div', this.hello)]
 }, 16 /* compiler generated slots */)
}
```

- 确保精确的组件级别依赖收集
- 避免不需要的父子关联更新

静态内容提取

Template

```
<div>
  <span class="foo">
    Static
  </span>
  <span>
    {{ dynamic }}
  </span>
</div>
```

Compiler output

```
const __static1 = h('span', {
  class: 'foo'
}, 'static')

render() {
  return h('div', [
    __static1,
    h('span', this.dynamic)
  ])
}
```

- 跳过整块静态内容的更新
- 即使静态内容在列表中被重复也可以生效

静态属性提取

Template

```
<div id="foo" class="bar">
  {{ text }}
</div>
```

Compiler output

```
const __props1 = {
 id: 'foo',
 class: 'bar'
}

render() {
 return h('div', __props1, this.text)
}
```

- 跳过单个元素的 patch，但依然处理子元素

内联事件函数提取

Template

```
<Comp @event="count++"/>
```

Compiler output

```
import { getBoundMethod } from 'vue'

function __fn1 () {
  this.count++
}

render() {
 return h(Comp, {
   onEvent: getBoundMethod(__fn1, this)
 })
}
```

- 避免因每次渲染生成新的内联函数而导致的不必要子组件更新

基于 Proxy 的新数据监听系统

全语言特性支持 + 更好的性能

- 对象属性增添 ／ 删除
- 数组 index / length 更改
- Map, Set, WeakMap, WeakSet
- Classes

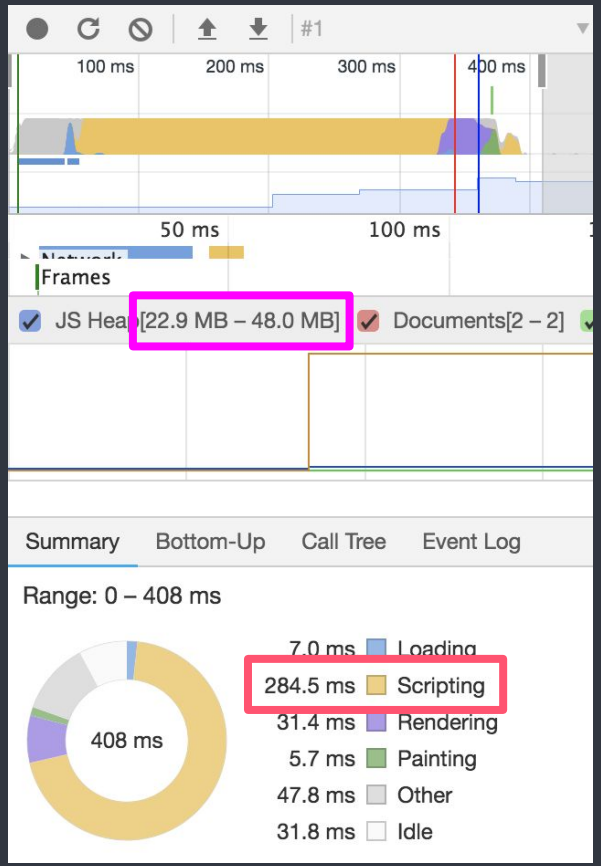
利用 Proxy 减少组件实例初始化开销

Bye Object.defineProperty!

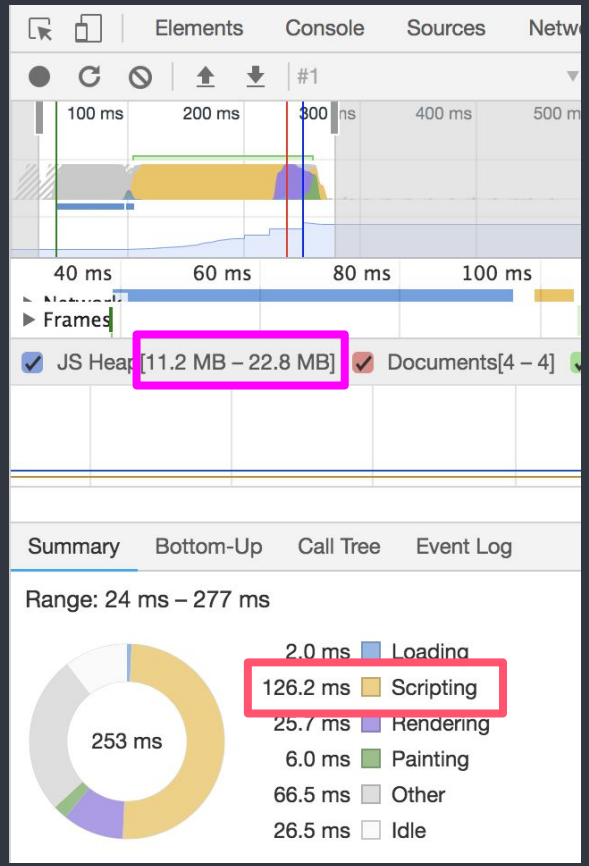
组件实例初始化
加快达 100%

速度加倍
内存占用减半

v2.5



v3.0-proto



- 渲染 3000 个带状态的组件实例

更小

便于 Tree-shaking 的代码结构

- 内置组件 (keep-alive, transition...)
- 指令的运行时 helper (v-model, v-for...)
- 各种工具函数 (asyncComponent, mixins, memoize...)

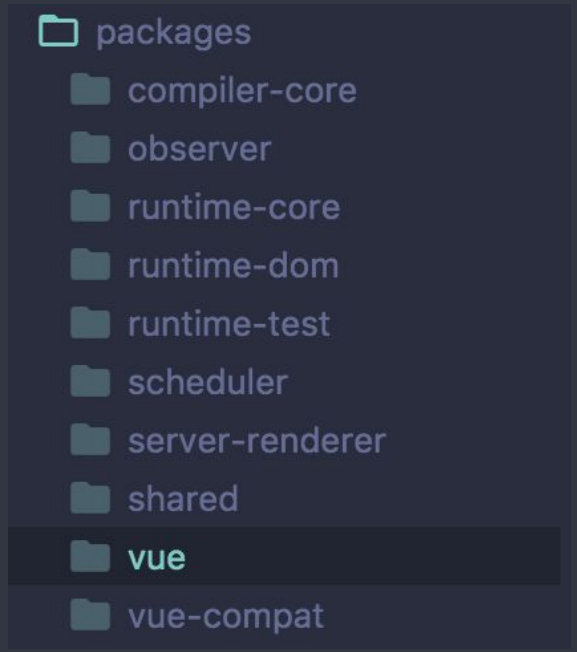
新的最小核心运行时: ~10kb gzipped

更易于维护

Flow -> TypeScript

（不影响用户代码）

内部模块解耦


packages
compiler-core
observer
runtime-core
runtime-dom
runtime-test
scheduler
server-renderer
shared
vue
vue-compat

编译器重构

- 插件化设计
- 带位置信息的 parser (source maps!)
- 为更好的 IDE 工具链铺路

更好的多端渲染支持

Custom Renderer API

```
import { createRenderer } from '@vue/runtime-core'

const { render } = createRenderer({
 nodeOps,
 patchData
})
```

新功能

响应式数据监听 API

```
import { observable, effect } from 'vue'

const state = observable({
 count: 0
})

effect(() => {
 console.log(`count is: ${state.count}`)
}) // count is: 0

state.count++ // count is: 1
```

轻松排查组件更新的触发原因

```
const Comp = {
  render(props) {
    return h('div', props.count)
  },
  renderTriggered(event) {
    debugger
  }
}
```

Demo

更好的 TypeScript 支持
包括原生的 Class API 和 TSX

```tsx
interface HelloProps {
  text: string
}

class Hello extends Component<HelloProps> {
  count = 0

  render() {
    return <div>
      {this.count}
      {this.$props.text}
    </div>
  }
}
```

Demo

更好的警告信息

- 组件堆栈包含函数式组件
- 可以直接在警告信息中查看组件的 props
- 在更多的警告中提供组件堆栈信息

Demo

Experimental Hooks API

作为一种逻辑复用机制，大概率取代 mixins

Demo

Experimental
Time Slicing Support

Demo

关于 IE...

会有一个专门的版本
在 IE11 中自动降级为旧的 getter／setter 机制
并对 IE 中不支持的用法给出警告

Thank you!