A

Brief History of CSS

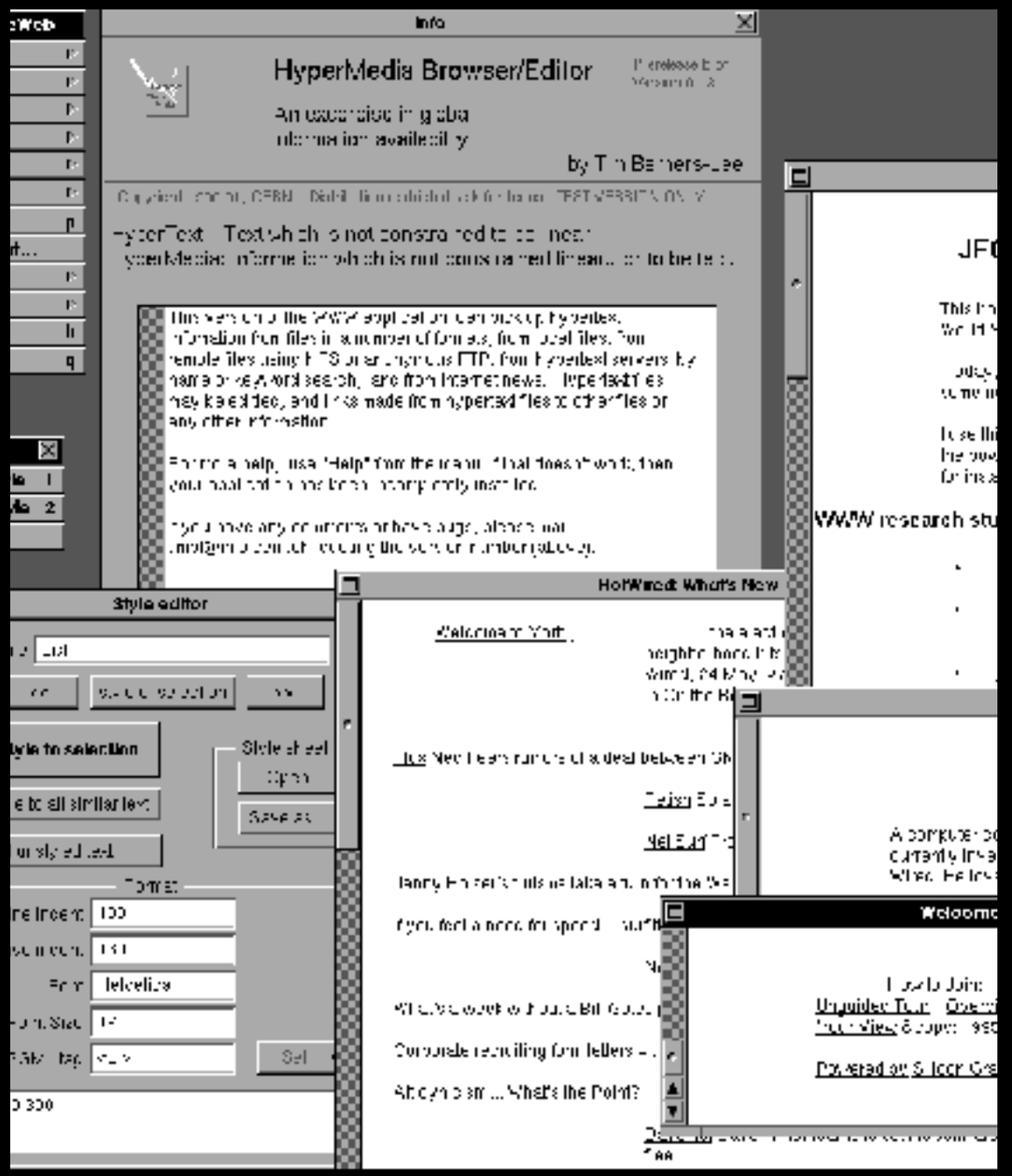
何时何地谁创造了CSS?

- 1994
- 在欧洲核子中心工作(CERN)的
- Håkon Wium Lie

Håkon Wium Lie, 12 December 1995

为什么能诞生？

- 结构和排版分离，是 HTML 诞生之初就有的设计目标
- Tim Berners-Lee，WWW 的发明人，HTML 的主要设计者之一，在编写 NeXT 上的浏览器（右图）时，已经在用自己的样式表了
- 作者为文档添加样式的需求变得强烈，甚至有人用图片代替页面
- 1994 前后，大量的样式表提案开始出现



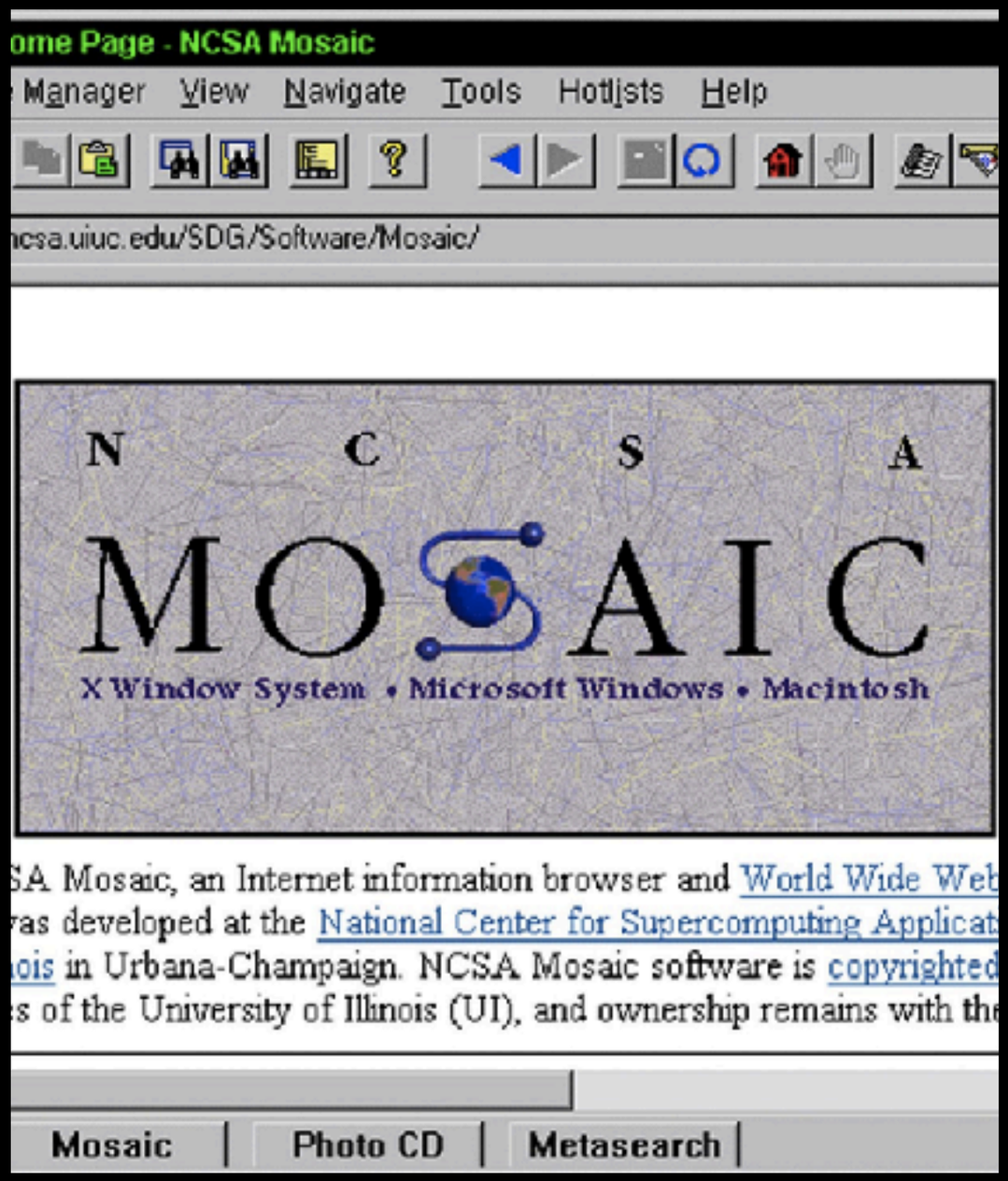
Tim’s editor

为什么是CSS?

- 也许是根正苗红：CERN
- CSS 因为 cascading 最特殊，可使文档的样式由作者、读者、显示设备和浏览器共同决定
- Bert Bos，Argo 浏览器作者的鼎力支持
- 微软的看好和 IE 在第一次浏览器大战中的胜出
- CSS 的设计目标是简单易用

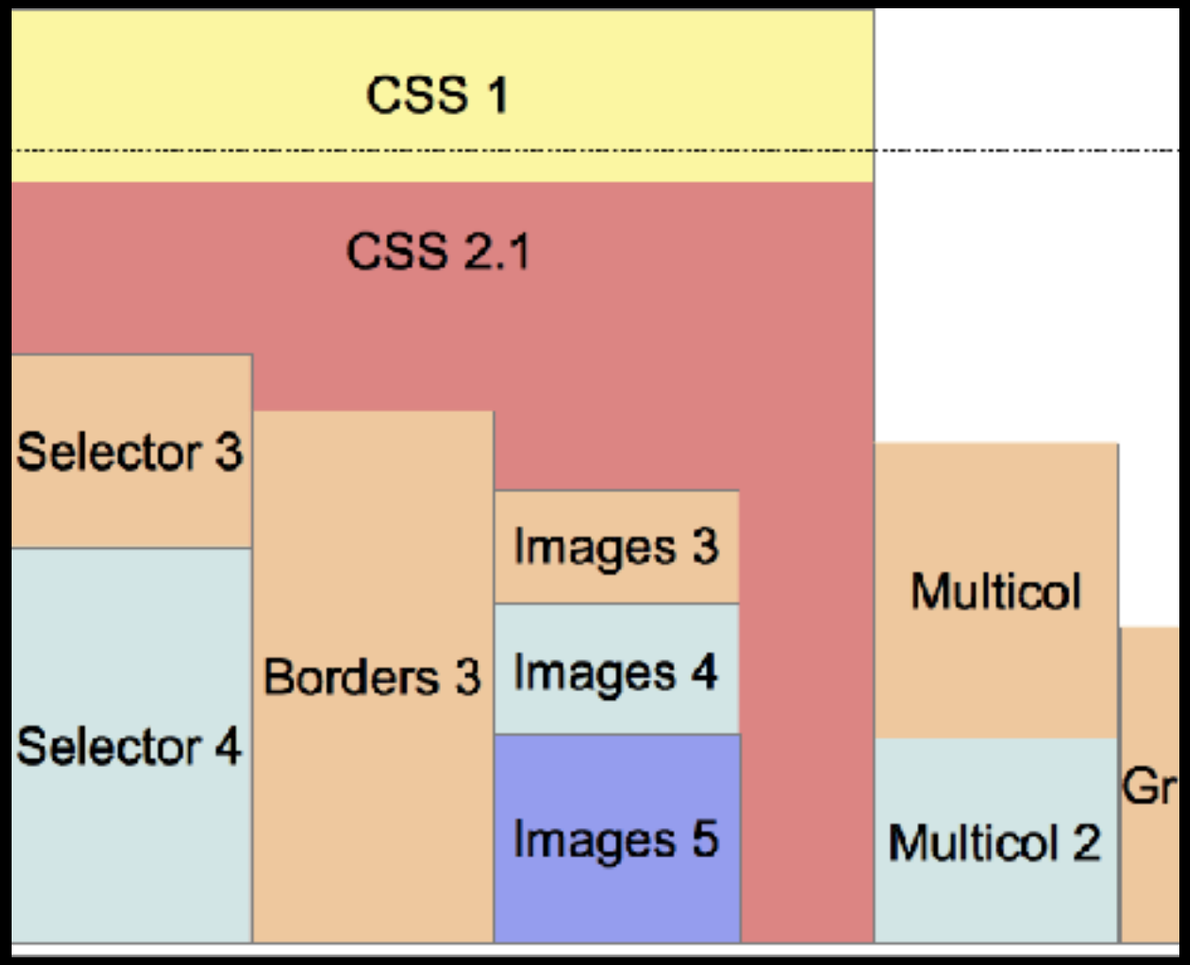
Netscape的情非得已

- Mosaic 的主程序员 Marc Andreessen 认为 HTML 不需要强大的样式系统
- 他后来是 Netscape 的联合创始人
- CSS 风头正盛
- IE 出于一部分要和 Netscape Navigator (alias Mozilla) 竞争的原因，站队 CSS
- Netscape 被动迎战开始支持 CSS
- Netscape 机智地将 CSS 转为 JSSS，以便随时绕过 CSS
- 第一次浏览器大战以微软 IE 胜利告终
- Mozilla 涅槃重生与 IE 开始第二轮大战，竟然比 IE 对 CSS 支持得更好，初心已忘 :)



CSS1 CSS2 CSS3

- CSS1 已废弃，CSS2.1 是现在的标准，CSS3 是还在路上的标准
- CSS2 的新特性
- CSS3 各个模块的进度



常见的CSS3模块

- Selectors
- Text
- Cascading and Inheritance
- Box Model
- Color
- Backgrounds and Borders
- Fonts
- Transforms
- Transitions
- Animations
- Layouts
  - Multi-column Layout
  - Flexible Box Layout
  - Grid Layout

CSS在成长

- 1994-10，Håkon 发触 CSS 提议的邮件
- 1996-8，支持 CSS 的 IE3 发布
- 1996-12，W3C CSS level 1 推荐标准发布
- 1998-5，W3C CSS level 2 推荐标准发布
- 1999-1，CSS1 Test Suite，Box Acid Test
- 1999-6，CSS3 草案发布
- 2005-4，Acid2 Test
- 2005-10，Sass 预处理器
- 2009-8，LESS 预处理器

还让 HTML4.0(1997-12)
废弃了“错误”的标签

- s/strike
- u
- font/basefont
- center
- ……

开发社区也在成长

- 1994-10，Håkon 发触 CSS 提议的邮件
- 1996-8，支持 CSS 的 IE3 发布
- 1996-12，W3C CSS level 1 推荐标准发布
- 1998-5，W3C CSS level 2 推荐标准发布
- 1999-1，CSS1 Test Suite，Box Acid Test
- 1999-6，CSS3 草案发布
- 2005-4，Acid2 Test
- 2005-10，Sass 预处理器
- **2009-5，Node.js 发布**
- 2009-8，LESS 预处理器
- 2006-8，jQuery
- **2009-5，Node.js 发布**
- 2010-7，Knockout.js
- 2010-10，Angular.js
- 2010-10，Backbone.js
- 2011-8，Bootstrap
- 2011-12，Ember.js
- 2012，Meteor.js
- 2013-3，React.js
- 2013-5，Polymer.js
- 2014，Vue.js

大时代带来的大挑战

- 2005 以前，简单的 CSS 可以轻松处理简单的问题
- 2005 ~ 2013，需要各种“最佳实践”在应对复杂的 web
- 2014 以后，“最佳实践”面临新的考验

常见的“最佳实践”

- 可维护性
  - reset
  - 使用一套命名规则 (OOCSS等)
  - 模块化
  - 合并选择和利用继承来减少代码
  - 单向 margin
  - ……
- 性能
  - 高性能的选择器
  - 尽量避免性能昂贵的属性
  - 雪碧图
  - 压缩
  - ……

MaintainableCSS https://maintainablecss.com/

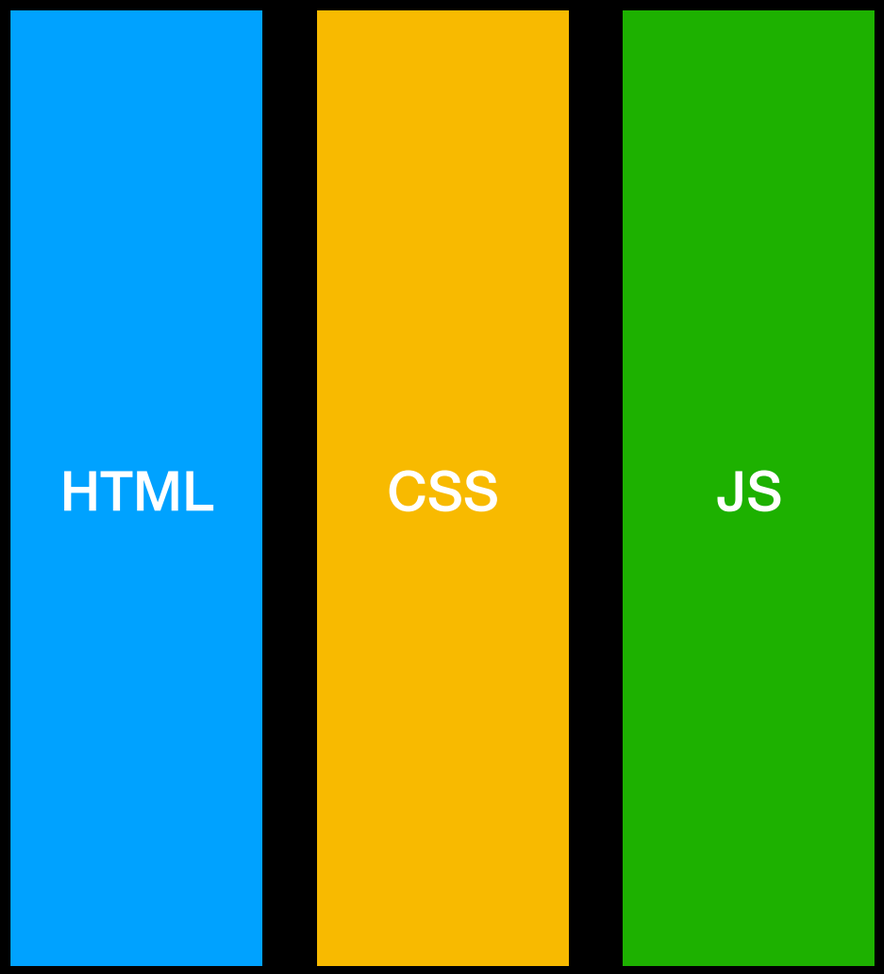
CSS规模扩大的问题

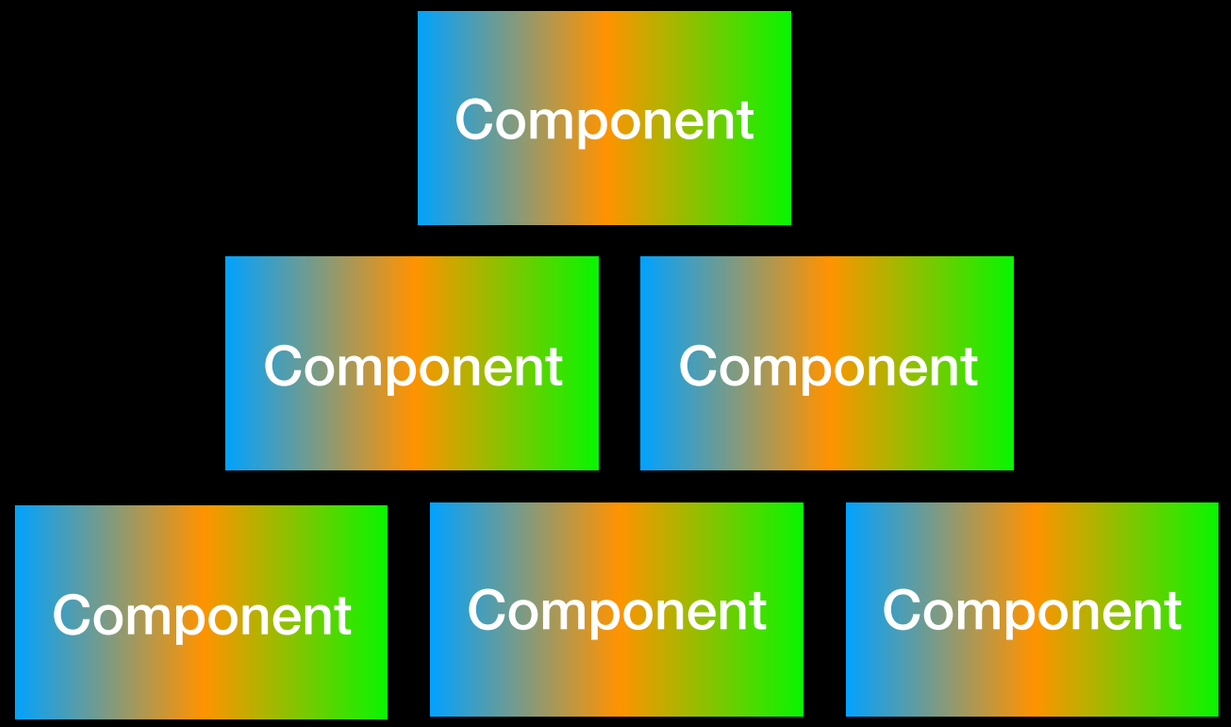
- 全局作用域
- 过度的选择器（为了避免 cascade 刻意使选择器更复杂）
- 重构难题

组件风暴

- 2004，Gmail 发布
- 2005，Google Maps 发布
- webapp 成为了不可忽视的需求
- Angular Directive，React，Vue 带来了 component
- component 是解决 webapp 工程问题的重要手段，我们更关注 webapp 和 component

Document and Webapp





新的难题

- Problems with CSS at scale
  1. Global Namespace
  2. Dependencies
  3. Dead Code Elimination
  4. Minification
  5. Sharing Constants
  6. Non-deterministic Resolution
  7. Isolation

Before we get to the crazy JS part, I'm going to go over all the issues we've been facing when trying to use CSS at scale and how we worked around them.

When I'm saying at scale, it means in a codebase with hundreds of developers that are committing code everyday and where most of them are not front-end

2014-10，Vjeux 的 CSS in JS 演说，彻底激发了 CSS in JS 的造轮子热潮。https://speakerdeck.com/vjeux/react-css-in-js

CSS in JS

- http://cssinjs.org/
- https://github.com/threepointone/glamor
- https://github.com/Khan/aphrodite
- https://github.com/styled-components/styled-components
- https://github.com/rofrischmann/fela
- https://github.com/emotion-js/emotion
- ……

CSS in JS 为什么能诞生?

- JSX 在 React 社区的深入人心。
- 动态化 CSS 的需求日益增长。
- view = f(state) 思想的流行，启发了 style = f(state)。
- 大统一，用 JS 的方式重新解决传统的 CSS “难题”。
- 真正隔离的组件化。

JSX 让 style=f(state) 变得简单

```jsx
function Button({ disabled }) {
  const style = {
    backgroundColor: disabled ? 'grey' : 'blue',
    color: 'white',
    border: 'none',
    padding: '6px 12px',
  }

  return <button style={style}>OK</button>
}
```

JS 使样式更加动态

```
function Button({ disabled, themeTextColor }) {
  const style = {
    backgroundColor: disabled ? 'grey' : 'blue',
    color: themeTextColor,
    border: 'none',
    padding: '6px 12px',
  }
  return <button style={style}>OK</button>
}

const ButtonWithTheme = withTheme(Button);

...

<ThemeProvider>
  <ButtonWithTheme />
</ThemeProvider>
```

JS 使样式更加动态

```
.width-1-24 { width: 8.33% }
…
.width-24-24 { width: 100% }

<div className="width-10-24"></div>
```

因为把计算静态化，CSS 文件可能会变大。

```
const widthOf24 = span => `${span / 24}%`
<div style={{ width: widthOf24(10) }}></div>
```

需要一些计算。

Just JS

- CSS 本质上是特定领域的编程语言（DSL）
- Sass/LESS 本质上是新的 DSL
- 如果你的工作更多是在“特定领域”之外，DSL 可能会给你带来很多限制
- Saas 的提供的语言特性，JS 也可以轻松地提供
- 没有“新语言问题”，即没有因引入新语言带来学习成本和语言缺陷

{less}

Learn Less In 10 Minutes

(or less)

Learn SASS

in 15 minutes

CSS in JS ≠ inline style

- 用 JSX 书写 inline style 很方便，但没有 CSS selector 的话，会失去很多 CSS 的功能。

- CSS in JS 不是颠覆 CSS，他本质上仍是 CSS，是由 JS 动态生成的 CSS。

- CSS in JS 是 CSS 生成的一种实现。

styled-components

```
import styled, { ThemeProvider } from 'styled-components';

const Button = styled.button`
  background-color: ${props.disabled ? 'grey' : 'blue'};
  color: ${props.theme.textColor};
  border: none;
  padding: 6px 12px;
`;

Button.defaultProps = {
  theme: {
    textColor: 'white',
  }
}

const theme = {
  textColor: 'white',
}

<ThemeProvider theme={theme}>
  <Button>OK</Button>
</ThemeProvider>
```

“There are only two hard things in Computer Science: cache invalidation and naming things.”

–Phil Karlton

CSS in JS 的优点

- You want to style the element, not class it.
- CSS 和 JS 常量共享，比如为 canvas 内容设置主题。
- 隔离性，你甚至难以写 CSS 来 hack 组件的样式。
- CSS 的继承很强大，不过有时会引入意外的 bug，一些 CSS in JS 库会设置 important 来避免。
- 单一的依赖树，统统并入 JS 的依赖系统。

不选择 CSS in JS 也没问题

- 也许你不喜欢或不需要动态性，静态的 CSS 就好。
- Saas/LESS 同样保证了 CSS 的可维护性。
- 用 BEM 命名规范解可以决隔离性。
- 或者 CSS module 来解决。
- 更习惯现有的 CSS 工具链。

cssday2017 CSS Reset

- https://github.com/nienkedekker/cssday2017#css-reset

- 也许他们设计的 CSS 不太适合用来写 webapp :)

End