# CSS 工程化与调试之路

——李耀东

# OUTLINE

- 什么是工程化
- 工程化所解决的问题
- CSS 调试如何适应当前的工程化

# 工程化的定义

将前端看成一项系统工程，从提高效率、解决问题的角度而抽象出来的一套方案。

- 分离结构、样式、行为
- 自动化代码构建、压缩、发布
- 使用各种框架、库来解决一些问题
- ...


不要给我说什么
底层原理、框架内核！
老夫敲代码就是
一把 梭 ！
复制！
粘贴！
拿起键盘就是
干！

# 刀耕火种

```
<html lang="en">
  <head>
    <link href="css/index.css" rel="stylesheet"/>
    <style>
      html,body {
        margin: 0;
        padding: 0;
      }
      .root {
        background-color: #f0f;
      }
    </style>
  </head>
  <body>
    <div class="root"></div>
  </body>
</html>
```

# 问题在哪

- 代码越来越多
- 作用域无法控制
- 跨页面的样式难以复用
- 难以响应需求的快速变化
- 没有底层架构，团队成员合作困难
- 浏览器初始化样式、前缀等兼容问题
- ...

# 作用域控制

```
// page-a.css
.menu-button {
  color: #fff;
  background-color: #f0f;
}

// page-b.css
.menu-button {
  background-color: #fff;
}
```

# 项目规范

- 文件、文件夹命名、分层
- 代码风格

# 想要解决的问题

- 基本的人员协作
- 代码风格的一致
- 基本的架构支持
- 应对需求变化/重构

# 文件、文件夹命名

```
|- src/css
   | -- settings.scss 变量、配置
   | -- reset.scss 浏览器初始化样式
   | -- mixin.scss 共用函数
   | -- basic.scss 基本边距、全局控件
   | -- shared/components-*.scss 复用组件
   | -- pages/*.scss 页面逻辑
```

- 底层：settings/reset/basic
- 复用：mixin/shared components/element
- 应用：pages(layout)/*.scss

# 代码风格

- 缩进
- 分号
- 空格
- 换行
- 命名
- 属性声明顺序
- 颜色
- ...

# 效果/问题

- 一定程度上让项目更有条理
- 仍然解决不了代码复用、平台支持
- 全靠自觉

# 预处理器

给予 CSS 额外的能力

- 文件切分
- 模块化复用
- 变量、运算、函数
- ...

# 想要解决的问题

- 作用域控制
- 文件的切分
- 灵活的配置
- 代码复用
- ...

# 代表

# 功能

- import
- variable
- nesting
- mixin/extends
- function/colors/math
- if/else/loop

# 效果/问题

- 弥补 CSS 的不足 (import, variable)
- 提升了开发效率(nesting)
- 一定程度上解决了代码复用(mixin/extends/function)
- 嵌套导致打包出来的代码大小未知(import, nesting, mixin)
- 增加了工具链的复杂度

# 框架/UI库

- Bootstrap
- Foundation
- Semantic UI
- UI Toolkit
- ...

# 想要解决的问题

- 浏览器兼容、一致性
- 增加复用，提升开发效率
- 约束代码风格
- 降低编写样式的门槛

# 功能

- ->reset/normalize
- typography/icon/element/component
- grid/responsive classes
- ...

# 效果/问题

- 降低浏览器兼容性成本
- 提升了开发效率
- 提供了架构支持
- 维护、升级、迁移成本变高
- CSS 体积增大

# 设计思想

通过对代码组织方式的的思考，形成固有的模式，统一开发体验，解决因项目变大而带来的问题。

# 想要解决的问题

- 作用域控制
- 代码的复用、扩展
- 及时相应需求的变化
- 良好的代码架构

# OOCSS

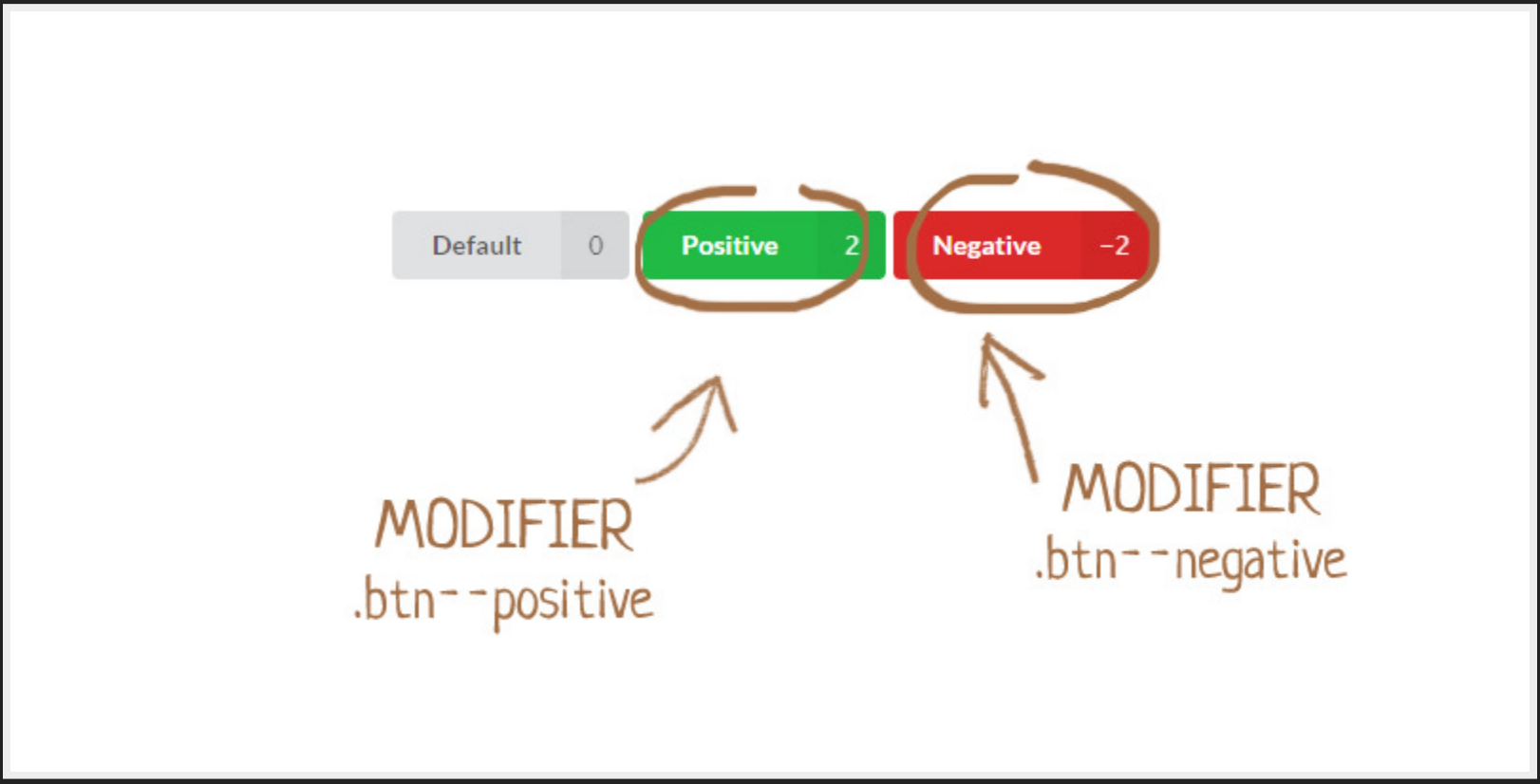
- Separate structure and skin

```
.box {
  width: 100px;
  margin: 10px 20px;
}
.box-shadow {
  color: #000;
  border: 1px solid #f0f;
}
```

- Separate container and content

```
.container {
  width: 1000px;
  overflow: hidden;
}
.content {
  text-align: center;
  padding: 10px;
}
```

# BEM




```
Block: .btn
Element .btn__description
Modifier .btn--positive
```

# 效果/问题

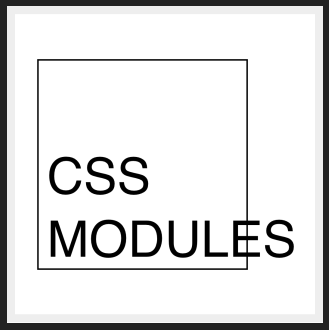
- 结合代码规范有效地解决了团队协作问题
- 良好的代码风格有利于代码复用及扩展
- 通过命名避免过长的嵌套，减小了 CSS 体积
- BEM 类名过长，写起来很冗余
- 仍然通过约定，无法强制避免作用域冲突

# CSS 模块化

通过对 CSS 进行模块化处理以达到更好的作用域控制以及复用的目的。

# 想要解决的问题

- 作用域控制
- 按需载入
- 压缩文件大小
- 复用




# -> webpack demo

```
html: <h1 class="title">An example heading</h1>

style.css: .title { background-color: red; }

js:
import styles from "./styles.css";
element.innerHTML =
  `<h1 class="${styles.title}">
     An example heading
   </h1>`;

output:
<h1 class="_styles__title_309571057">An example heading</h1>
```

# WEB COMPONENTS

->CSS scoping specification

```
<dom-module id="x-foo">
  <template class="sign-up-template">
      <style>
        button {
          background-color: #f0f;
        }
      </style>
      <input type="text" id="username" placeholder="username">
      <input type="password" id="password" placeholder="password">
      <button id="btn">Sign Up!</button>
  </template>
</dom-module>
```

# 效果/问题

- 通过动态类名的方法解决了作用域问题
- 通过绑定HTML实现按需加载
- 一定程度上实现了跨平台(React Native)
- JS 更加灵活易于实现复杂特性(更换主题)
- 需要更复杂的构建工具以及 JS 框架支持
- 学习成本以及编辑器支持

# 构建优化

通过相关构建工具，提升开发效率，优化打包、发布流程。

# 想要解决的问题

- 代码的压缩、自动发布
- 浏览器兼容
- 代码风格不统一
- 页面手动刷新太慢
- 代码调试

- PostCSS: CSS 后处理器构建平台
- CSSNext: 使用最新的 CSS 特性而无需考虑兼容性
- Autoprefixer: 浏览器前缀自动补全
- Stylelint: CSS 代码风格检查
- Livereload/Hot Module Replacement: 浏览器自动刷新
- SourceMap: 源代码定位
- ...

# 效果/问题

- 较好地解决了大工程的自动化构建
- 提升了开发效率
- 增加了工具链的复杂度

# 调试

## 调试工具能做到的

- CSS 权重优先级是什么
- CSS 最终是否应用到了 DOM 上
- 这行 CSS 的性能损耗是多少

## 不能做到的

- 为什么这行 CSS 应用上了但却不是我想要的效果

# 常用功能

- computed panel
- box model
- hover state
- color picker/convert to different format
- sourcemap
- rendered font
- mobile device simulator (responsive)
- animations panel
- rendering panel
- coverage
- ->performance panel

# 总结

- 工程化是前端项目复杂之后的必要措施
- 每项措施都有背后的原因，需要权衡
- 浏览器和规范正在不断完善，部分优化将无需特殊支持 (http2, web components, css variable)

# Q&A

# THANKS