02

OPEN ORIENTED

凹 凸 实 验 室

CSS变量

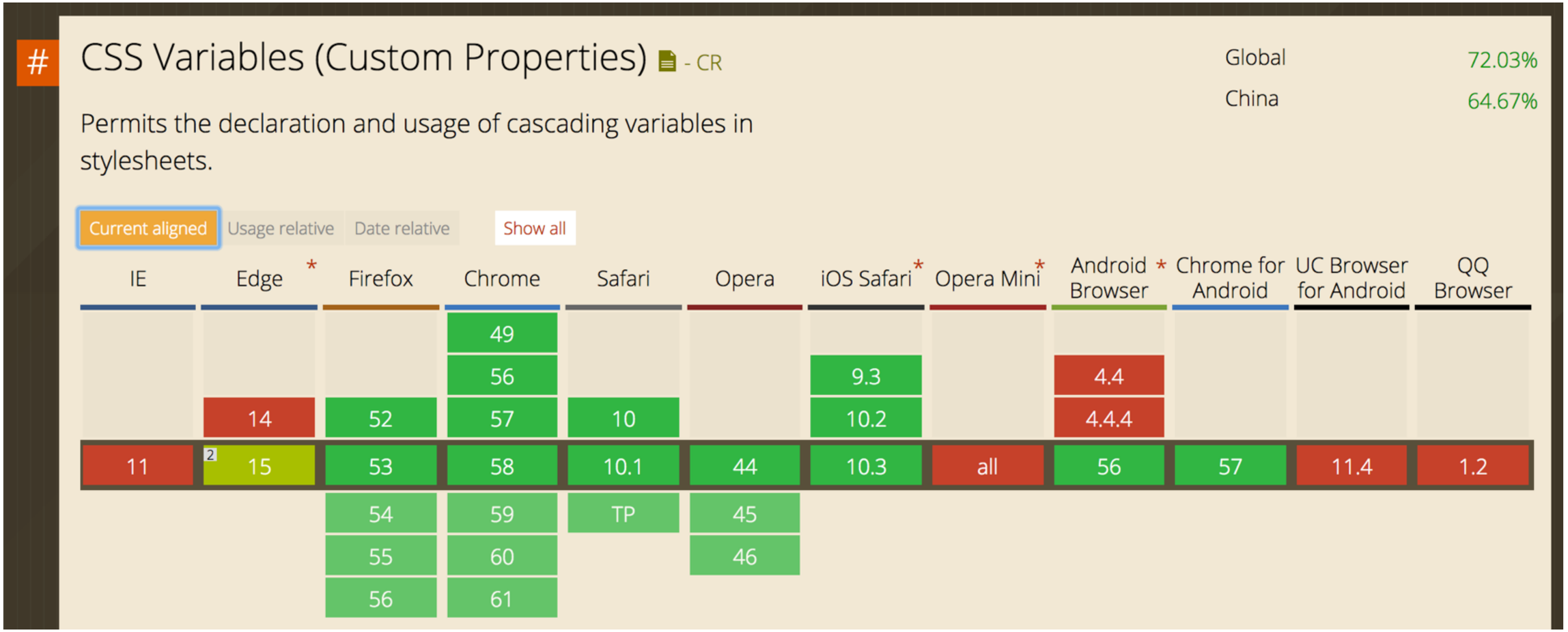
Tinglelaoo

Define Once，Use Everywhere

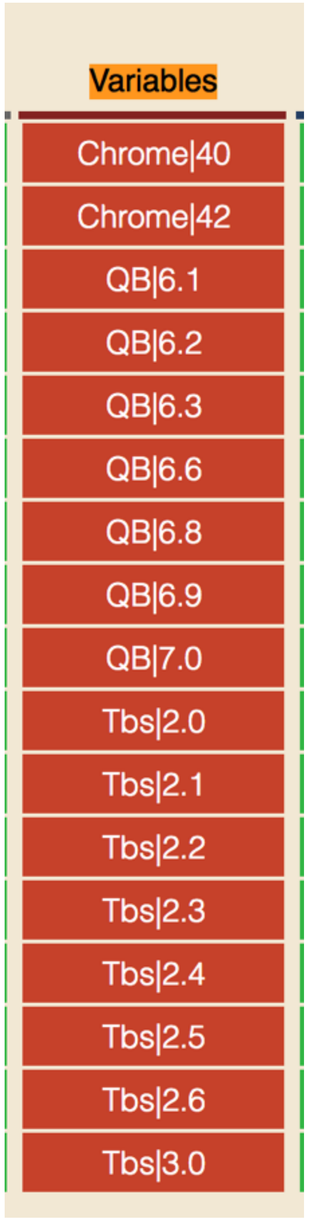
CSS自定义属性（也称为CSS变量），我们在Sass、LESS见识过，也使用频繁。

而现在W3C见识到它的强大，开始着手于这方面的支持工作，目前把CSS自定义属性的内容纳入了CSS4标准的草案阶段。

兼容性-Can I Use


#
CSS Variables (Custom Properties) - CR
Permits the declaration and usage of cascading variables in stylesheets.
Global 72.03%
China 64.67%
Current aligned
Usage relative
Date relative
Show all
IE
Edge *
Firefox
Chrome
Safari
Opera
iOS Safari *
Opera Mini *
Android Browser *
Chrome for Android
UC Browser for Android
QQ Browser
49
56
57
58
59
60
61
14
15
2
52
53
54
55
56
10
10.1
TP
44
45
46
9.3
10.2
10.3
all
4.4
4.4.4
56
57
11.4
1.2

兼容性－X5 Can I Use



定义


```css
.foo {
  --theme-color: blue;
  --fallback-color: red;
  --spacer-width: 8px;
  --favorite-number: 3;
  --greeting: "Hey, what's up?";
  --reusable-shadow: 0 3px 1px -2px rgba(0, 0, 0, 0.85);
}
```

命名规范：

- 以**“—”双横杠开头**，后面可以是数字[0–9]、字母[a–zA–Z]、下划线_和短横线–这些组合，甚至是中文也行，但不能包含$、[、^、(、%等字符。
- **大小写敏感**，––foo与––Foo代表不同的CSS变量。
- 只能在**声明块{ }**内定义。

使用


```
5   .mod_section {
6       background-color: var(--theme-color);
7   }
```

var() 函数仅用于解析CSS变量，接收第二个参数作为缺省值。

```
.mod_section {
    background-color: var(--theme-color,#ff3900);
}

.mod_section {
    background-color: var(--theme-color,var(--fallback-color));
}
```

作用域与级联

以**元素级别划分**的作用域，分全局作用域和局部作用域。

在 **:root 伪元素**内定义的CSS变量，可认为是全局变量，属于全局作用域内。

```
:root {
    --theme-color: blue;
}
```

在其它类名内定义的，是局部变量，属于局部作用域内，遵从样式的级联规则。

```
.mod_section {
    --theme-color: #e4393c;
    background-color: var(--theme-color);
}
```

作用域与级联规则

1.局部定义可覆盖全局变量的值

```css
:root {
  --main-color: #06c;
}

#foo h1 {
  color: var(--main-color); /* #06c */
}

#foo h2 {
  --main-color: #0cc;
  color: var(--main-color); /* #0cc */
}
```

作用域与级联规则

2.可带上'!important'

```
:root {
    --theme-color: #666 !important;
    --theme-color: blue;
}
.mod_section {
    background-color: var(--theme-color); /* #666 */
}
```

作用域与级联规则

3.解决依赖循环问题的依据

```
<one>
    <two>
        <three />
    </two>
</one>
```

```
one   { --foo: 10px; }
two   { --bar: calc(var(--foo) + 10px); }
three { --foo: calc(var(--bar) + 10px); }
```

其它注意事项－单位处理

注意单位，CSS变量解析后会默认跟随一个空格

```
.foo {
  --gap: 20;
  margin-top: var(--gap)px; //错误，解析为 margin-top: 20 px;
}
.foo {
  --gap: 20;
  margin-top: calc(var(--gap) * 1px); //正确，解析为margin-top: 20px;
}
```

其它注意事项－无效值

如果是无效值，那么会使用CSS属性的**默认值(initial value)**

或者是**继承值(inherit value)**

```
:root {
  --looks-valid: 20px;
}
p {
  background-color: red;
}
p {
  background-color: var(--looks-valid); //无效，解析为默认值transparent。
}
```

其它注意事项－使用限制

1.CSS自定义属性变量是不能用作CSS属性名称的

```
.foo {
    --source-attr: 'margin-top';
    var(--source-attr): 20px;  // 错误，不可用作CSS属性名称
}
```

2.不能用作图像等资源定位地址

```
.foo {
    --image-url: 'http://jdc.jd.com/img/290x290';
    background-image: url(var(--image-url)); // 错误，不可用作图像等资源定位地址
}
```

配合CSS calc()

calc() 函数常常被用于跨单位的计算

```
.calc {
    width: calc(100% - 16px);
}
```

结合calc() 函数，使得CSS变量显得更灵活、强大

```
:root {
  --base-size: 10px;
}

.mod_section {
  margin-bottom: calc(2 * var(--base-size));
}
```

与JavaScript桥接

其实，原生的CSS变量兼容性支持并不好，是不是不值得学习和使用呢?是不是应该抛弃而是直接用预处理器处理（Sass、Less）更好呢?

但，它带来了一种新的与JavaScript桥接的方式。除了现有的内联处理，JavaScript可以通过改变CSS变量来控制样式。

与JavaScript桥接

和**平时用 JS 操作元素任意的属性**一般，自定义属性也可以通过 **getPropertyValue** 和 **setProperty** 方法操作：

```
const styles = getComputedStyle(document.querySelector('.foo'));

// 获取自定义属性
const oldColor = styles.getPropertyValue('--color').trim();

// 设置自定义属性
foo.style.setProperty('--color', 'green');
```

使用场景－CSS、JavaScript解耦合

通常的做法是通过 JS 动态添加或移除 class，改变视觉效果。

```css
.button {
  position: relative;
  transform: scale(1);
}

.button.js-toggled {
  transform: scale(1.5);
}
```

```js
const button = document.querySelector('.button');

button.addEventListener('click', () => {
  button.classList.toggle('js-toggled');
});
```

但是，灵活性不够强。如果情况很复杂，可能还需要动态地传入一些值的时候，就需要内联样式、甚至于动态创建样式去处理。

使用场景－CSS、JavaScript解耦合

而通过CSS变量（自定义属性），就可以很好地优化。

```css
.container {
  position: relative;
  --clickX: 0;
  --clickY: 0;
}

.container > .auxElement {
  position: absolute;
  transform: translate(var(--clickX, 0), var(--clickY, 0));
}
```

```js
const container = document.querySelector('.container');

container.addEventListener('click', evt => {
  container.style.setProperty('--clickX', `${evt.clientX}px`);
  container.style.setProperty('--clickY', `${evt.clientY}px`);
});
```

使用场景－控制伪元素

通过CSS变量，也能控制伪元素的样式

```css
.container {
  position: relative;
  --clickX: 0;
  --clickY: 0;
}

.container::after {
  position: absolute;
  transform: translate(var(--clickX, 0), var(--clickY, 0));
}
```

参考文章

知乎专栏－CSS 自定义属性 — 使用篇

知乎专栏－CSS 自定义属性 — 基础篇

必须了解的CSS变量（var）

Google – CSS Variables: Why Should You Care?

CSS Custom Properties for Cascading Variables Module Level 1

THANKS

FOR YOUR WATCHING

02

OPEN ORIENTED

凹 凸 实 验 室