目录

[配置编译 2](#_Toc534446226)

[cli 2](#_Toc534446227)

[webpack 2](#_Toc534446228)

[注意 4](#_Toc534446229)

[变量 4](#_Toc534446230)

[基本语法 4](#_Toc534446231)

[与sass区别 5](#_Toc534446232)

[默认值 6](#_Toc534446233)

[calc()运算 6](#_Toc534446234)

[关键词 7](#_Toc534446235)

[all 7](#_Toc534446236)

[JavaScript控制自定义属性 7](#_Toc534446237)

[浏览器支持检测 8](#_Toc534446238)

[mixin 8](#_Toc534446239)

[其他 9](#_Toc534446240)

[媒体查询 9](#_Toc534446241)

[常用插件 9](#_Toc534446242)

[postcss-import 9](#_Toc534446243)

[cssnext 9](#_Toc534446244)

[autoprefixer 10](#_Toc534446245)

[颜色函数 10](#_Toc534446246)

[css-mqpacker 10](#_Toc534446247)

[cssnano 10](#_Toc534446248)

[cssnext 10](#_Toc534446249)

[color 10](#_Toc534446250)

postcss本身并没有任何语法标准

# 配置编译

## cli

postcss --use autoprefixer -c options.json -o main.css css/\*.css

## webpack

v1

var path = require('path');

module.exports = {

context: path.join(\_\_dirname, 'app'),

entry: './app',

output: {

path: path.join(\_\_dirname, 'dist'),

filename: 'bundle.js'

},

module: {

loaders: [

{

test: /\.css$/,

loader: "style-loader!css-loader!postcss-loader"

}

]

},

postcss: function () {

return [

require('postcss-import')({

onImport: function (files) {

files.forEach(this.addDependency)

}.bind(this)

}),

require('postcss-simple-vars')(),

require('postcss-focus')(),

require('autoprefixer')({

browsers: ['last 2 versions', 'IE > 8']

}),

require('postcss-reporter')({

clearMessages: true

})

]

},

target: 'web',

stats: false,

progress: true

}

}

v1迁移到v2

plugins: {  
**new webpack.LoaderOptionsPlugin({  
options: {  
postcss: function () {  
return [precss, autoprefixer];  
},**devServer: {  
contentBase: "./public", //本地服务器所加载的页面所在的目录  
colors: true, //终端中输出结果为彩色  
historyApiFallback: true, //不跳转  
inline: true //实时刷新  
}  
}  
})

**v2正式写法**

 rules: [{

  test: /\.css$/,

  use: [

**'style-loader'**, {

   loader: 'css-loader',

   options: {

   // modules: true // 设置css模块化,详情参考<https://github.com/css-modules/css-modules>

importLoaders: 1

   }

  }, {

   loader: 'postcss-loader',

   // 在这里进行配置，也可以在postcss.config.js中进行配置，详情参考<https://github.com/postcss/postcss-loader>

**options: {**

**plugins: function() {**

**return [**

**require('autoprefixer')**

**];**

**}**

**}**

  }

  ]

 },

**也可以在postcss.config.js中进行配置，**

module.exports = {

plugins: **[**

**require('precss'),**

**require('autoprefixer')**

**]**

}

## 注意

When postcss-loader is used standalone (without css-loader) don't use @import in your CSS, since this can lead to quite bloated bundles

在单独使用postcss（没有css-loader）时，不要使用@import。

# 变量

CSS自定义属性仅适用于属性值，他们不能被用于选择器，属性名称或媒体查询。

## 基本语法

未来的css变量标准

:root {

--white: #fff;

--grey: #1e1e1d;

--yellow: #ffad15;

--offwhite: #f8f8f8; ... }

a {

color: **var(--yellow)**;

}

:root表示匹配HTML中的<html>元素，具有最高的权重，类似的

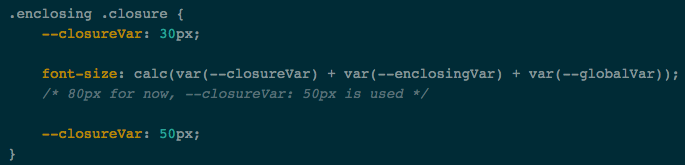
.enclosing{}

<div class="enclosing">

相当于sass在一个选择器内部申明变量

## 与sass区别

1



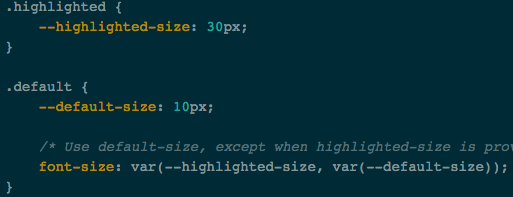
**如果你重新分配一个自定义属性的值，浏览器将重新计算cale()表达式所运用的变量**。

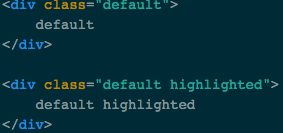
而在sass没有效果。

2

**CSS自定义属性可以理解DOM的结构和动态改变**。

Sass的计算和处理都是发生在编译的时候，当然，它不知道任何关于DOM结构，完全依赖于代码的结构。





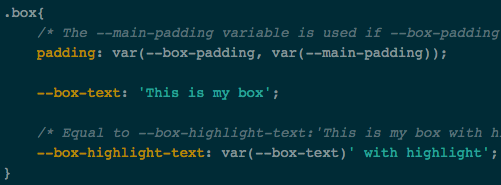
比较两个



## 默认值

var()函数可以提供一个默认值。如果你不能确定自定义属性是否被定义，可以通过这个方式提供一个值作为后备。

默认值本身也可以是一个变量



## calc()运算

计算,css3的标准

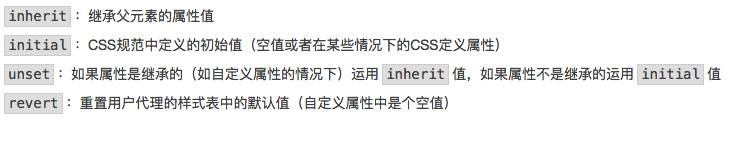
不能计算不带单位的值。



表达式中有“+”和“-”时，其前后必须要有空格，如"widht: calc(12%+5em)"这种没有空格的写法是错误的；

表达式中有“\*”和“/”时，其前后可以没有空格，但建议留有空格。

## 关键词



--animation: revert;

## all

重置所有的CSS属性。

.my-wonderful-clean-component{ **all: initial;** }

不幸的是，all关键字并不能重置所有自定义属性。现在正在进行讨论是否要添加前缀--，将重置所有CSS自定义属性。

## JavaScript控制自定义属性

以前，从CSS发送数据到JavaScript，我们不得不求助于一些技巧，通过JSON编写CSS，做为输出，然后通过JavaScript来阅读它。

现在，我们可以很容易的通过JavaScript来处理CSS变量，我们所知道的就是通过.getPropertyValue()和.setProperty()方法阅读和重写自定义属性。

element.style.setProperty(`--${varName}`, value);

getComputedStyle(element).getPropertyValue('--phone');

## 浏览器支持检测



# mixin

:root {

--danger-theme: {

color: white;

background-color: red;

};

}

.danger {

@apply --danger-theme;

}

# 其他

## 媒体查询

@custom-media --small-viewport (max-width: 30em);

/\* check out media queries ranges for a better syntax !\*/

@media (--small-viewport) {

/\* styles for small viewport \*/

}

注意两个括号

# 常用插件

## postcss-import

替代Sass样式中的@import功能。

@import './test.css';

cssnext包含这个？

## cssnext

把CSS未来特性编译成现在CSS语法特性。特别需要注意的是，这个插件和Sass和LESS都不一样。它提供的特性是CSS正在进行的规范（或者未来CSS将会具备的语法特性）。而且有一些特性已经得到了浏览器的支持。我使用cssnext来处理Sass不具备的一些特性。

包含autoprefixer，同时使用时cssnext会给出一个警告。

options:{

plugins:function(){

return [

require('postcss-cssnext')

]

}

}

## autoprefixer

cssnext包括了Autoprefixer插件

https://baike.baidu.com/item/Autoprefixer/22466404?fr=aladdin

## 颜色函数

cssnext包括了这个函数功能，

-darkerwhite: color(var(--offwhite) shade(20%));

现在cssnext编译出来的颜色结果是十六进制或rgba()值。当颜色函数在浏览器中运行时，编译将不会是必要的。颜色操作在运行时就可能发生。

## css-mqpacker

CSS媒体查询

## cssnano

优化代码

## precss

类似scss的插件

https://www.jianshu.com/p/9de7f190f408

和less之类功能重复，用less时不需要用这个，可能会有冲突，比如语法相同时。

# cssnext

## color

* [red( | green( | blue( | alpha( | a(] ['+' | '-']? [<number> | <percentage>] )
* [red( | green( | blue( | alpha( | a(] '\*' <percentage> )
* [hue( | h(] ['+' | '-' | '\*']? <angle> )
* [saturation( | s(] ['+' | '-' | '\*']? <percentage> )
* [lightness( | l(] ['+' | '-' | '\*']? <percentage> )
* [whiteness( | w(] ['+' | '-' | '\*']? <percentage> )
* [blackness( | b(] ['+' | '-' | '\*']? <percentage> )
* tint( <percentage> )
* shade( <percentage> )
* blend( <color> <percentage> [rgb | hsl | hwb]? )
* contrast( <percentage>? )

例子

color: color(red alpha(-10%));

color: color(red blackness(80%));

color(hsla(125, 50%, 50%, .4) saturation(+ 10%) w(- 20%));

# bug-fix

一般像正常的插件一样引用即可

## postcss-flexbugs-fixes

https://github.com/luisrudge/postcss-flexbugs-fixes