Stylus

- **Stylus 官网文档**
- **Important Warning:** 下面的笔记本不是完整文档, 只是一些常用语法的书写, 完整文档请参考官网文档.

Catalog

    - 1.28 JavaScript API
    - 1.29 SourceMaps (资源图)
    - 1.30 CSS3 extensions with NIB (Nib 下 CSS 扩展)
2. 在 Vue 组件中使用 stylus

New Words

- **interpolation [ɪn,tɚ˞pə'leʃən]{US} --n.插值; 插入; 填写**
    - linear interpolation. 线性插值.
- **operator ['ɒpəreɪtə]/['ɑpəretə˞] --n.[计]操作符, 运算符; 操作员; 经营者**
    - operator overloading 运算符重载.
    - Operator, I want to make a long distance call. 话务员, 我要打长途电话.
- **conditional [kən'dɪʃ(ə)n(ə)l] --adj.有条件的, 受约束的**
    - a conditional contract 附有条件的契约
    - conditional comment. 条件注释.
- **comment ['kɒment] --n.评论, 注释 --v.评论; 注释**
    - Comment is needless. 解释是不必要的.
    - add comments or explanations. 加注释或说明.
    - the top commenter, a bottom-pinner,
- **iteration [ˌɪtə'reɪʃn] --n.迭代, 重复, 反复**
    - Let's listen to the next iteration of this theme. 我们来听下一段主题.
    - So this process takes about four or five iterations. 这个过程需要四到五此反复.
    - Iterators are just objects with a specific interface designed for iteration. 迭代器是一种特殊对象, 它具有一些专门为迭代过程设计的专有接口.
- **executable ['eksikju:təbl] --adj.可执行的, 实行的**
    - C++ Executable C++ 可执行文件
- **introspection ['ɪntrə'spɛkʃən] --n.内省; 反省.**
    - And with this, they gained introspection. 有了这样的认知, 它们学会了自省.
- **partial ['pɑ:ʃ(ə)l] -- --adj.局部的, 部分的, [口]偏爱的**
    - partial success. 部分的成功.
    - partial to detective novels. 偏爱侦探小说
- **range [reɪn(d)ʒ] --n.范围, 山脉 v.排列**
    - This is outside the range of our study. 这不属于我们研究的范围.
- **disambiguation ['dɪsæm,bɪgjʊ'eɪʃən] --n.消除歧义; 解疑**
- **invoke [ɪn'vəʊk] vt.调用; 祈求;**

    - Immediately-Invoked Function Expression. IIFE 立即调用函数表达式.
    - By now I can no longer invoke that [excuse [ɪk'skjuːz/]. 现在我不能再用这个借口了.
- **saturate ['sætʃərɪt] --vt.使饱和; 使浸透; 渗透**
    - The rain had saturated our clothes by the time we got home. 等我们抵家时, 衣服被雨水淋得湿透了.
    - saturate condition. 饱和条件; 饱和状态
- **desaturate [diː'sætʃəreɪt] --vt.降低饱和度**
- **saturation [sætʃə'reɪʃ(ə)n]/['sætʃə'reʃən] --n.饱和; 饱和度;**
    - saturation pressure. 饱和压力.
    - Though China has a large market, for us, it is getting to the saturation point. 对我们来说, 尽管国内市场很大, 但已接近饱和.
- **blend [blɛnd] --vt.混合,混杂. --vi.混和,调和[with]. --n.混和**
    - blend(vt) milk and cream(together). 混和牛奶和奶油.
    - Sea and sky seemed to blend. 大海和蓝天似乎相融合.
- **complementary [kɒmplɪ'ment(ə)rɪ] --adj.补充的, 互补的**
    - complementary set 补集
    - complementary angles 余角, 互余角
    - complementary function 余函数
- **hue [hju] --n.色调; 色彩; 色度**
    - a change in hue. 色调上的变化
    - a cold [warm] hue. 冷 [暖] 色
    - The diamond shone with every hue under the sun. 金刚石在阳光下放出五颜六色的光芒.
- **spin [spɪn] --vt&vi.纺<纱>; 旋转. --n.转动**
    - spin(vt) cotton into thread. 把棉纺成纱
    - spin(vt) a coin. 抛转钱币.
- **equivalent [ɪ'kwɪvələnt] --n.当量; 相等. --adj.相等的; 相当的**
    - These two words are equivalent(adj) in meaning. 这两个字意义相等.
    - What is $3 equivalent(adj) to in Japanese yen? 三美元相当于多少日元呢?
    - There is no exact English equivalent(n) for this Chinese expression. 这个中文辞句在英文中没有完全相等的辞句.
- **tint [tɪnt] --vt.着色. --n.色彩; 染色**
    - autumnal(n) tints. 秋色
    - (a) green of [with] a blue tint(n). 带着淡蓝色的绿色.
    - The sunset tinted(vt) the hills. 落日把山丘染上了淡淡的蓝色.
- **shade [ʃeɪd] --n.阴; 阴影; 深浅.**
    - There is not much shade there. 那里阴凉的地方不多.
    - The tree gives a pleasant shade. 这棵树提供了一块舒适阴凉的地方.
- **luminosity [luːmɪ'nɒsətɪ]/[ˌlʊmə'nɑsəti] --n.亮度; 发光; 光明**

- **luminance ['lumɪnəns] --n.亮度; 明亮度**
  - hight luminance. 高亮度
- **contrast ['kɒntrɑːst]/[/'kɑntræst/] --n.对比; 对照. --vt.比较,对照. --vi.形成对照**
  - the contrast(n) between light and shade. 明暗的对照.
  - by contrast. 对照之下.
  - What a contrast(n) between them! 他们之间的差别多大!
  - contrast(vt) two things. 使两物相对比.
- **delimiter [dɪ'lɪmɪtə-] --n.分隔符; 定界符**
  - field demimiter. 字段分隔符.
  - boundary delimiter. 边界分隔符.
- **literal ['lɪt(ə)r(ə)l] --adj.文字(上)的, 字面的, 逐字的**
  - a literal error. 错字, 排错字
  - in the literal sense of the word. 按那字的字面意义.
  - a literal translation. 逐字翻译, 直译.

前置知识 (Pre-knowledge)

1. CSS 选择符 (Selectors)

- (1) 元素选择符
  - 通配选择符 (*)
  - 类型选择符 (E, 例如: div, p, h1, ...)
  - ID 选择符 (E#id)
  - 类选择符 (E.class)
- (2) 关系选择符
  - 包含选择符 (E F)
  - 子选择符 (E > F)
  - 相邻选择符 (E + F)
  - 兄弟选择符 (E ~ F)
- (3) 属性选择符
  - E[att] : 更多示例见 **CSS 参考手册**
  - E[att = "val"]
  - E[att ~= "val"]
  - E[att ^= "val"]
  - E[att $= "val"]
  - E[att *= "val"]
  - E[att |= "val"]

- (4) 伪类选择符
- (5) 伪对象选择符

Content

0. Stylus 安装

0.1 安装 Stylus

- 使用 npm 来安装

```
  # - 全局安装
  # - Mac 是使用 nvm 安装的 node, 所以 node 默认的 npm 包管理器也是在 nvm
  #   目录下:
/Users/WANG/.nvm/versions/node/v13.11.0/lib/node_modules/stylus/bin/stylus
  npm install stylus -g
```

- 使用 yarn 安装

```
  yarn global add stylus
```

0.2 用法

- `0.1` 使用 npm 全局安装 stylus 后, 我们在进入 stylus 文件所在的文件夹中, 直接运行下面的命令来编译 stylus 文件为对应的 css 文件:

  ```
  # -o 是 out 缩写
  stylus xxx.styl -o xxx.css
  ```

  那么现在就遇到另一个问题了, 如果我们每次写完 stylus 文件, 保存后都要手动执行上面的命令来重新编译文件, 那这就太不友好了, 所以 stylus 提供了解决方法, 就是在上面的基础命令上添加 `-w` 即可:

  ```
  # -w (watch) 监听
  stylus -w xxx.styl -o xxx.css
  ```

- Tip: stylus 写法说明: stylus 的写法是根据缩进来识别的, 所以:
  - 冒号可有可无
  - 分号可有可无
  - 逗号可有可无
  - 括号可有可无

1. Stylus 基础语法

1.1 Selector (选择器)

- (1) 缩进
  - Stylus 语法是类 `Python` 的 (即: 基于缩进). 空格很重要, 所以使用 **缩进** 和 **突出** 来替代 `{` 和 `}`. 如下所示:

    ```
    body
        color white
    ```

    编译成 CSS 为 (While compiles to):

    ```
    body {
        color: #fff;
    }
    ```

    如果你喜欢, 可以使用冒号来分隔属性和值:

    ```
    body
        color: white;
    ```

- (2) 规则集 (Rule Sets)
  - 和 CSS 一样, stylus 允许通过 `,` 逗号分隔, 一次为多个选择器定义属性:

    ```
    textarea, input
        border 1px solid #ccc
    ```

    也可以通过换行书写来实现:

    ```
    textarea
    input
        border 1px solid #ccc
    ```

    最终都会编译为:

    ```
    textarea,
    input {
        border: 1px solid #ccc;
    }
    ```

    这个规则有一个例外就是外观类似于属性的选择器. 例如, 下面这个 `foo bar baz` 可能是一个属性或一个选择器:

    ```
    foo bar baz
    > input
        border 1px solid
    ```

    由于上述原因, 我们可以在末尾添加一个逗号:

    ```
    foo bar baz,
    form input,
    > a
        border 1px solid
    ```

Notice: 上面两段 stylus 代码我都不能在 stylus 文件中正确编译, 这个 `>` 符号是什么意思? 还有类似于属性的选择器我懂, 但是我还是不明白这里要说的是什么意思! (如果有大神看懂了, 欢迎告诉我.)

- (3) 父级引用 (Parent Reference)
  - `&` 符号代表父级选择器. 下面的例子中, `textarea` 和 `input` 两个选择器的伪类 `:hover` 都会改变 color 属性.

```
textarea
input
    color #a7a7a7
    &:hover
        color #000
```

编译为:

```
textarea,
input {
    color: #a7a7a7;
}
textarea:hover,
input:hover {
    color: #000;
}
```

下面的例子, 在 `混入(mixin)` (tip: 后面会讲到) 中使用父级引用为 IE 浏览器的元素添加了一个简单的 `2px` 的边框:

```
box-shadow()
    -webkit-box-shadow arguments
    -moz-box-shadow arguments
    box-shadow arguments
    html.ie8 &,
    html.ie7 &,
    html.ie6 &
        border 2px solid arguments[length(arguments) - 1]
body
    #login
        box-shadow 1px 1px 3px #eee
```

编译为:

```
body #login {
    -webkit-box-shadow: 1px 1px 3px #eee;
    -moz-box-shadow: 1px 1px 3px #eee;
    box-shadow: 1px 1px 3px #eee;
}
html.ie8 body #login,
html.ie7 body #login,
html.ie6 body #login {
    border: 2px solid #eee;
}
```

如果需要在选择器中单纯第使用 `&` 符号, 不适用其父级引用的功能, 可以通过转义符 `\` 来转义:

```
.foo[title*='\&']
    color black
```

编译后为:

```
.foo[title*='&'] {
    color: #000;
}
```

- (4) 局部引用 (Partial Reference)
  - `^[N]` 出现在选择器的任意位置, `N` 是一个数字, 表示部分引用. 部分引用的工作方式类似于父级引用, 但是父级引用包含着整个选择器, 而部分引用仅包含着嵌套选择器的第 `N` 级, 所以你可以单独引用嵌套中的某个级别. `^[0]` 将获取到第一级选择器, `^[1]` 获取第二级选择器元素, 以此类推. 例如

```
.foo
    &_bar
        width: 10px
        ^[0]:hover &
            width: 20px
```

编译成 CSS 为:

```
/* - .foo_bar 默认的 width 为 10px */
.foo_bar {
    width: 10px;
}
/* - 当我们的鼠标移动到父级的 class 为 .foo 的元素上时,
     .foo_bar 的宽度变为 20px */
.foo:hover .foo_bar {
    width: 20px;
}
```

~~负值将从结尾开始开始计算. 因此 `^[-1]` 将返回 `&` 链式前的最后一个选择器.~~ .....

- (5) 局部引用中的范围 (Ranges in partial Reference)
  - 略...
  - 理论行来说, 省略的内容在开发中几乎用不到, 又加之时间有限, 不再书写下面的内容, 如果想看本章节("选择器")的汉语翻译, 请移步这篇 **文章**, 英文文档请移步 **Stylus 官网**
- (6) 初始化引用 (Initial Reference)
  - `~/` : 略
- (7) 相对引用 (Relative Reference)
  - `../` : 略
- (8) 根引用 (Root Reference)
  - `/` : 略

- (9) selector() 内置函数
  - 略...
- (10) 多参数值的 selector 内置函数 (Multiple values for `selector()` bif)
  - 略...
- (11) selectors() 内置函数
  - 略...
- (12) 消除歧义 (Disambiguation)
  - 像 `margin -n` 这样的表达式中 `-n` 可能会被解释为一个减法运算, 也可能解释成一个一元运算符, 为了避免这种分歧, 需要用括号包裹住表达式:

    ```
    pad(n)
        margin (-n)

    body
        pad(5px)
    ```

    编译为:

    ```
    body {
        margin: -5px
    }
    ```

    但是仅仅在函数中会这样(因为函数用返回值同时扮演着混合和回调).

    下面这个例子就是正常的 (和上面一样的结果):

    ```
    body
        margin -5px
    ```

    如果有 Stylus 无法处理的属性值, 那么 `unquote` 会帮到你:

    ```
    filter unquote('progid: DXImageTransform.Microsoft.BasicImage(rotation=1)')
    ```

    编译为:

    ```
    .box {
        filter: progid:DXImageTransform.Microsoft.BasicImage(rotation=1);
    }
    ```

    现在都 2020 年了, 请忽略上面的 `unquote` (结束引用) 的语法.

1.2 Variable (变量)

- (1) 变量: 我们可以将表达式赋值给变量, 然后在整个样式表中使用变量:

  ```
  font-size = 14px

  body
      font font-size Arial, sans-serif
  ```

编译为:

```
body {
    font: 14px Arial, sans-serif;
}
```

变量甚至可以包含一个表达式列表:

```
font-size = 14px
font-stack = "Lucida Grande", Arial, sans-serif

body
    font font-size font-stack
```

编译为:

```
body {
    font: 14px "Lucida Grande", Arial, sans-serif;
}
```

标识符(就是指: 变量、函数、属性的名字, 或者函数的参数) 还可以包含 `$` 符号. 例如

```
$font-size-normal = 14px
body {
    font: $font-size-normal sans-serif;
}
```

- (2) 属性查找: Stylus 的另一个功能是不需要分配值给变量就可以定义引用属性. 下面是一个使元素垂直居中对齐(tip: 典型的方法是使用百分比和 margin 负值, 如下) 的示例:

```
#logo
    position: absolute
    top: 50%
    left: 50%
    width: w = 150px
    height: h = 80px
    margin-left: -(w / 2)
    margin-top: -(h / 2)
```

无需定义变量 `w` 和 `h`, 我们可以仅仅通过前置 `@` 字符在属性名前来访问该属性名对应的值:

```
#logo
    position: absolute
    top: 50%
    left: 50%
    width: 150px
    height: 80px
    margin-left: -(@width / 2)
    margin-top: -(@height / 2)
```

编译为:

```
#logo {
    position: absolute;
    top: 50%;
    left: 50%;
    width: 150px;
    height: 80px;
    margin-left: -75px;
    margin-top: -40px;
}
```

另外的使用案例是基于其他属有条件地定义属性. 在下面这个例子中, 我们默认指定 `z-index` 的值为 1, 但是, 这只有在 `z-index` 之前未指定的时候才会使用默认值:

```
position()
    position: arguments
    z-index: 1 unless @z-index

#my-logo
    z-index: 20
    position: absolute
#your-logo
    position: absolute
```

编译为 CSS 为:

```
#my-logo {
    z-index: 20;
    position: absolute;
}
#your-logo {
    position: absolute;
    z-index: 1;
}
```

`@` 符号后的 "属性" 会 "向上冒泡" 查找堆栈直到被发现, 或者没找到时返回 null. 下面这个例子, `@color` 被解析成 blue:

```
body
    color: red
    ul
        li
        color: blue
        a
            background-color: @color
```

1.3 Interpolation (插值)

- (1) 插值: Stylus 支持插入值, 通过使用 `{}` 中包裹的表达式, 表达式的结果将会变成定义的一部分. 例如, `-webkit-{'border' + '-radius'}` 等同于 `-webkit-border-radius`.

  一个很好的例子就是对于私有前缀额外扩展属性.

```
vendor(prop, args)
    -webkit-{prop}: args
    -moz-{prop}: args
    {prop}: args
border-radius()
    vendor('border-radius', arguments)
box-shadow()
    vendor('box-shadow', arguments)
button
    box-shadow: 1px 1px 2px 3px #fafafa
    border-radius: 1px 2px/3px 4px
```

编译为 CSS 为:

```
button {
    -webkit-box-shadow: 1px 1px 2px 3px #fafafa;
    -moz-box-shadow: 1px 1px 2px 3px #fafafa;
    box-shadow: 1px 1px 2px 3px #fafafa;
    -webkit-border-radius: 1px 2px/3px 4px;
    -moz-border-radius: 1px 2px/3px 4px;
    border-radius: 1px 2px/3px 4px;

    /* - 上面的 box-shadow 等价于下面 4 行 */
    /* border-top-left-radius: 1px 3px;
    border-top-right-radius: 2px 4px;
    border-bottom-right-radius: 1px 3px;
    border-bottom-left-radius: 2px 4px; */
}
```

- (2) 选择器插值: 插值也可以在选择器中运用到. 例如, 我可以插入值去指定一个表格中前 5 行的高属性值, 向下面这样:

```
table
    for row in 1 2 3 4 5
        tr:nth-child({row})
            height:10px*row
```

生成为:

```
table tr:nth-child(1) {
    height: 10px;
}
table tr:nth-child(2) {
    height: 20px;
}
table tr:nth-child(3) {
    height: 30px;
}
table tr:nth-child(4) {
    height: 40px;
}
table tr:nth-child(5) {
```

```
    height: 50px;
}
```

你也可以把多个选择器当做一个字符串放在一个变量中, 去插入到它们应该在的地方:

```
mySelector = '#foo, #bar, .bar'
{mySelector}
    background: #000
```

编译为:

```
#foo,
#bar,
.baz {
    background: #000;
}
```

1.4 Operators (运算符)

- (1) 一元运算符: `!, not, -, + ~`
- (2) 二元运算符: 下标运算符允许我们可以通过索引获取表达式内部值(从 0 开始), 负值可以从表达式中最后的元素开始.

```
// - 总的来说或 这种语法很奇怪, 很明显就可以看出来 list 变量引用的是一个数组,
//   既然这样的话, 为什么不写成 list = [1, 2, ,3] 这样难道不更明显吗?
list = 1 2 3
.lists
    z-index: list[0]    // => 1

.lists-test2
    z-index: list[-1]    // => 3. 3(list.length) + (-1) = 2 => list[2] = 3
```

编译为:

```
.lists {
    z-index: 1;
}
.lists-test2 {
    z-index: 3;
}
```

- (3) 范围运算符 `.. / ...` : 范围运算符包含: (开放)范围运算符: `..` 和 排除(闭合)范围运算符 `...` 去扩展表达式:

```
1..5    // => 1 2 3 4 5
1...5   // => 1 2 3 4
5..1    // => 5 4 3 2 1
```

- (4) 指数运算符 `**`

```
2 ** 8   // => 256
```

- (5) 相等和关系运算符: `==, !=, >=, <=, >, <`
- (6) 真实性
- (7) 逻辑运算符: `&&(逻辑与) / ||(逻辑或)`
- (8) 存在运算符: `in`

```
nums = 1 2 3
1 in nums   // => true

5 in nums   // => false
```

- (9) 条件赋值 `?= / :=` : 条件赋值运算符 `?=` 别名 `:=` , 让我们定义变量不需要破坏原来的值(如果存在的话). 这个操作运算符可以扩展为三元中的一个 `is defined` 的二元操作. 例如下面是相等的:

```
color := white    // <=>  color ?= white
color = color is defined ? color : white
```

但是下面这种情况表示(当变量已经定义):

```
color = white
color ?= black  // - 表示: 如果 color 已经存在, 就把 black 赋值给 color
.box
    background-color: color // => black
```

- (10) 实例检查: `is a` : stylus 提供了一个二元操作符 `is a` 用于类型检测.

```
15 is a 'unit'      // => true
#fff is a 'rgba'    // => true
15 is a 'rgba'      // => false
```

或者我们可以使用内置函数 `type()`

```
type(#fff) == 'rgba'    // => true
```

- (11) 变量是否定义: `is` : 这个运算符仅仅检查我们的定义的变量是否有值. 例如:

```
foo is defined      // => false 因为 foo 未定义

foo = 15px
foo is defined      // => true

#fff is defined     // => 'invalid "is defined" check on non-variable #fff' (无效的 "定义", 请检查非变量 #fff)
```

  - **Notice**: 这里的笔记并不是 Stylus 官网 `Operators(运算符)` 的全部内容, 完整内容请移步官网(tip: 中文文档仍然是英文的 -_-)

1.5 Functions (方法/函数)

Stylus 具有强大的内置于语言层面的函数定义. 函数定义与 `混入(mixins)` 一致, 但是却可以返回值.

Notice: Stylus 的函数返回值前面默认是不写 `return` 的, 但是由于个人习惯, 下面的笔记内部全部都添加了 `return` .

- (1) 返回值: 一个简单的例子: 创建一个两数相加的函数:

```
add(a, b)
    return a + b
```

我们可以将上述函数用在条件表达式中, 作为属性值等等.

```
body
    padding: add(10px, 5)   // - 注意, 第二个参数不带单位, 默认使用第一个参数的单位
```

编译为:

```
body {
    padding: 15px;
}
```

- (2) 默认参数: 和 ES6 语法一样, 也支持默认参数, 例如:

```
add(a, b = a)
    return a + b
add(10, 5)      // => 15
add(10)         // => 20
```

同样和 ES6 一样, 也支持使用函数调用作为默认值:

```
// - tip: unit() 为内置函数. 参数(1) 为数值, 参数(2) 为接收单位. 后面会讲到.
add(a, b = unit(a, px))
    return a + b

.default-parameters
    // - tip: 此处参数 10 如果想带单位也是可以的, 不过应该要和上面 unit()
    //   中的一致.
    padding: add(10)
```

- (3) 命名参数: 函数可以接受命名参数, 这可以免去记忆参数顺序的麻烦, 或者说是提供了代码的可读性. 例如:

```
subtract(a, b)
    return a - b
.named-parameter
    padding: subtract(b: 10, a: 25)
```

编译为:

```
.named-parameter {
    padding: 15;
}
```

- (4) 函数体: 我们把 `add()` 函数做进一步的扩展. 通过内置 `unit()` 函数把所有参数的数值单位都变成 `px` . 因为它为每个参数重新赋值了, 并且每个参数都提供了字符串标识数值单位类型. 因此, 可以无视单位换算的内部机制.

```
add(a, b = a)
    a = unit(a, px)
    b = unit(b, px)
    return a + b

.fun-body
    // - Q: 不知道 Stylus 内部是如何实现单位之间的互相转换的,
    //   不同单位转换为 `px` 会根据不用的父级元素作参考, 是个很繁杂的工作...
    font-size: add(15%, 10deg)
```

编译后为:

```
.fun-body {
    font-size: 25px;
}
```

- (5) 多个返回值: Stylus 函数可以返回多个值, 就像给一个变量赋予多个值一样. 例如下面就是一个有效的变量赋值:

```
sizes = 15px 10px
sizes[0]        // => 15px
```

类似的, 我们可以在函数中返回多个值:

```
sizes()
    return 15px 10px
sizes()[0]      // => 15px
```

当返回值是标识符是个小小的例外. 例如, 下面的代码对于 Stylus 来说看上去好像是一个属性赋值 (但是没有赋值操作符):

```
swap(a, b)
    b a
```

为了避免歧义, 我们可以使用括号, 或者 `return` 关键字:

```
swap(a, b)
    (b, a)
swap(a, b)
    return b a
```

- (6) 条件 用法示例

```
compare(a, b)
    if a > b
        return higher
    else if a < b
        return lower
```

```
    else
        return qual

// - 用法
compare(5, 2)   // => higher
compare(1, 5)   // => lower
compare(10, 10) // => equal

// - Q: 问题这个示例在哪里可以用到?
```

- (7) 别名. 例如:

```
add(a, b = a)
    return a + b

plus = add

plus(1, 2)      // => 3
```

- (8) 函数作为变量使用: 和 JS 语法一样, 函数也可以被作为参数传递(高阶函数是指至少满足下列条件之一的函数: `(1) 函数可以作为参数被传递.``(2) 函数可以作为返回值输出.`). 如下, `invoke()` 函数能够接收一个函数作为参数, 因此我们可以将 `add()` 或 `sub()` 函数作为参数传递给 `invoke()`:

```
add(a, b)
    return a + b
sub(a, b)
    return a - b
invoke(a, b, fn)
    return fn(a, b)
.fun
    padding: invoke(5, 10, add)
    margin: invoke(5, 10, sub)
```

编译为:

```
.fun {
    padding: 15;
    margin: -5;
}
```

- (9) 匿名函数: 利用 `@(){}` 语法可以在需要的地方使用匿名函数. 下面展示了如何利用匿名函数创建一个自定义的 `sort()` 函数:

```
sort(list, fn = null)
    // default sort function
    if fn == null
        // - 这里的意思如果没有传入 fn 函数, 那么在此处利用 `@(){}` 方式,
        //   手动来调用, 调用的代码在下面嵌套的 for 循环内部.
        fn = @(a, b) {
            a > b
        }
```

```
    // bubble sort
    for $i in 1..length(light) - 1
        for $j in 0..$i - 1
            if fn(list[$i], list[$j])
                $temp = list[$i]
                list[$i] = list[$j]
                list[$j] = $temp
    return list

sort('e', 'c', 'f', 'a', 'b', 'd')  // => 'a', 'b'', 'c', 'd', 'e', 'f'

sort(5 3 6 1 2 4, @(a, b){
    a < b
})      // => 6 5 4 3 2 1
```

Notice: 由于无法自行测试这些函数, 所以我觉得用处应该不大.

- (10) 参数: 和 JS 相同. 给一个示例:

```
sum()
    n = 0
    for num in arguments
        n = n + num
sum(1, 2, 3, 4, 5)      // => 15
```

- (11) Hash 实例 : 略

1.6 Keyword Arguments (关键字参数)

- Q: 不知道干啥用的...
- Stylus 支持关键字参数或 "kwargs". 这样就能够通过其关联的参数名来引用参数.

  以下示例在功能上是等价的. 然而, 我们可以 在参数列表中的任何位置放置关键字参数. 未设置关键字的参数将被用来填充其余尚未填充的参数.

```
body {
    color: rgba(255, 200, 100, 0.5);
    color: rgba(red: 255, green: 200, blue: 100, alpha: 0.5);
    color: rgba(alpha: 0.5, blue: 100, red: 255, 200);
    color: rgba(alpha: 0.5, blue: 100, 255, 200);
}
```

  转换为:

```
body {
    color: rgba(255,200,100,0.5);
    color: rgba(255,200,100,0.5);
    color: rgba(255,200,100,0.5);
    color: rgba(255,200,100,0.5);
}
```

  若要查看函数或混合(mixin) 所能接受的参数, 请使用 `p()` 函数:

```
p(rgba)

// - 输出 inspect: rgba(red, green, blue, alpha)
```

1.7 Built-in Functions(内置函数)

WARNING: 这个章节的笔记并没有做完, 因为内置函数实在太多了, 而且也不知道这些到底会不会用得到, 所有函数的列表已经列出, 如果将来有用自己查找对应函数即可!

- (1) Color Functions(颜色函数)
  - (1.1) `red(color[, value])` : 返回给定颜色中的红色比重, 或将红色比重值设置为可选的第二个参数.

    ```
    red(#c00)           // => 204
    red(#000, 255)      // => #f00
    ```

  - (1.2) `green(color[, value])` : 大致意思和 (1.1) 相同.

    ```
    green(#0c0)         // => 204
    green(#000, 2555)   // => #0f0
    ```

  - (1.3) `blue(color[, value])` : 同上. 示例: 略.
  - (1.4) `alpha(color[, value])` : 返回给定 color 中的透明度比重, 或者将透明度比重设置为可选的第二个参数.

    ```
    alpha(#fff)                     // => 1
    alpha(rgba(0, 0, 0, 0.3))       // => 0.3
    alpha(#fff, 0.5)                // => rgba(255, 255, 255, 0.5)
    ```

  - (1.5) `dark(color)` : 检查 color 值是否是深色.

    ```
    dark(black)                     // => true
    dark(#005716)                   // => true
    dark(white)                     // => false
    ```

  - (1.6) `light(color)` : 检查 color 值是否是亮色.
  - (1.7) `hue(color[, value])` : 返回给定颜色值的色调, 或设置色调值为可选的第二个参数值.

    ```
    hue(hsl(50deg, 100%, 80%))      // => 50deg
    hue(#00c,90deg)                 // => #6c0
    ```

  - (1.8) `saturation(color[, value])` : 返回给定颜色的饱和度. 解释同上.

    ```
    saturation(hsl(50deg, 100%, 80%))   // => 100%
    saturation(#00c, 50%)               // => #339
    ```

  - (1.9) `lightness(color[, value])` : 返回给定颜色的亮度.

```
lightness(hsl(50deg, 100%, 80%))    // => 80%
lightness(#00c, 80%)                // => #99f
```

- (1.10) `hsla(color | h, s, l, a)` : 转换给定的颜色值到 `HSLA` 模式, 或者使用 `h, s, l, a` 的组合值.

```
hsla(10deg, 50%, 30%, 0.5)          // => HSLA
hsla(#ffcc00)                       // => HSLA
```

- (1.11) `hsl(color | h, s, l)` : 转换给定的颜色值到 `HSLA` 模式, 或者使用 `h, s, l, a` 的组合值. (tip: 和 (1.10) 类似, 只是少了 `alpha` 即透明度)

```
hsl(10, 50, 30)         // => HSLA
hsl(#ffcc00)            // => HSLA
```

- (1.12) `rgba(color | r, g, b, a)` : 从 `r, g, b, a` 通道返回 `RGBA` , 或根据所提供颜色来调整透明度.

```
rgba(255, 0, 0, 0.5)    // => rgba(255, 0, 0, 0.5)
rgba(255, 0, 0, 1)      // => #ff0000
rgba(#ffcc00, 0.5)      // => rgba(255, 204, 0, 0.5)
```

另外, stylus 支持 `#rgba` 以及 `#rrggbbaa` 符号

```
#fc08           // => rgba(255, 204, 0, 0.5)
#ffcc00ee       // => rgba(255, 204, 0, 0.9)
```

- (1.13) `rgb(color | r, g, b)` : 从 `r, g, b,` 通道返回 `RGBA` , 或者投射成一个 `RGBA` 节点.

```
rgb(255, 204, 0)        // => #ffcc00
rgb(#fff)               // => #fff
```

- (1.14) `blend(top[, bottom])` : 使用常规混合将给定的顶部颜色混合在 底部颜色上, `bottom` 参数是可选的, 默认是 `#fff` . 例如:

```
blend(rgba(#fff, 0.5), #000)            // => #808080
blend(rgba(#ffde00, .42), #19c261)      // => #7ace38
blend(rgba(lime, 0.5), rgba(red, 0.25)) // => rgba(128, 128, 0, 0.625)
```

- (1.15) `lighten(color, amount)` : 按给定数量(amount) 减轻给定的颜色值 (即: 把 color 变亮). 这个函数对单位敏感, 例如, 它支持百分比, 示例如下:

```
lighten(#2c2c2c, 30)   // => #787878
lighten(#2c2c2c, 30%)  // => #393939
```

- (1.16) `darken(color, amount)` : 按给定数量加深给定的颜色值(即: 把颜色加深). 这个函数对单位敏感, 也是支持百分比, 示例如下:

```
darken(#D62828, 30)     // => #551010
darken(#D62828, 30%)    // => #961c1c
```

- (1.17) `desaturate(color, amount)` : 按给定的数量减轻颜色的饱和度.

```
desaturate(#f00, 40%)   // => #c33
```

- (1.18) `saturate(color, amount)` : 按给定的数量增加颜色的饱和度.

```
saturate(#c33, 40%)     // => #f00
```

- (1.19) `complement(color)` : 给出指定颜色的互补色. 相当于旋转色相 180 度.

```
complement(#fd0cc7)     // => #0cfd42
```

- (1.20) `invert(color)` : 颜色反转. 红绿蓝颜色反转, 透明度不变.

```
invert(#d62828)     // => #29d7d7
```

- (1.21) `spin(color, amount)` : 按给出的数量旋转给定颜色的色相.

```
spin(#ff0000, 90deg)        // => #80ff00
```

- (1.22) `grayscale(color)` : 给出与给定颜色等效的灰度. 相当于去饱和度(颜色, 100%).

```
grayscale(#fd0cc7)          // => #0cfd42
```

- (1.23) `mix(color1, color2[, amount])` : 由给定的量混合 2 种颜色. 量是可选的, 其默认值为 50%.

```
mix(#000, #fff, 30%)        // => #b2b2b2
```

- (1.24) `tint(color, amount)` : 将给定的颜色与白色混合.

```
tint(#fd0cc7, 66%)          // => #feaceb
```

- (1.25) `shade(color, amount)` 将给定的颜色与黑色混合

```
shade(#fd0cc7, 66%)         // => #560443
```

- (1.26) `luminosity(color)` : 返回给定色彩的相对亮度

```
luminosity(white)       // => 1
luminosity(#000)        // => 0
luminosity(red)         // => 0.2126
```

- (1.27) `contrast(top[, bottom])` : 返回对象顶部和底部颜色之间的的对比度, 根据 `Lea Verou` 的 "对比度" 脚本. 略.....
- (1.28) `transparentify(top[, bottom, alpha])` : 透明度有关. 略...

- (2) Fath Functions(路径函数)
  - (2.1) `basename(path[, ext])` : 返回路径的 basename(基本名). 第二个参数是可选的, 作用为移除拓展名.

```
basename('images/foo.png')  // => "foo.png"
basename('images/foo.png', '.png')  // "foo"
```

- (2.2) `dirname(path)` : 返回路径的目录名.

```
dirname('images/foo.png')   // => "images"
```

- (2.3) `extname(path)` : 返回路径的文件扩展名, 包括点.

```
extname('images/foo.png')   // => ".png"
```

- (2.4) `pathjoin(...)` : 执行拼接路径

```
pathjoin('images', 'foo.png')   // => "images/foo.png"

path = 'images/foo.png'
ext = extname(path)
pathjoin(dirname(path), basename(path, ext) + _thumb + ext)
// => "images/foo_thumb.png"
```

- (3) List/Hash Functions(列表/哈希 函数)
  - (3.1) `push(expr, args...)` : 把第二个及其以后的参数(arguments) 传给第一个表达式(expression). 别名为 `append()`

```
nums = 1 2
// - tip: 实际上就是推入项到数组. 这里不得不说一下 stylus 写法的问题,
//   上面的 nums = 1 2 直接写成 nums = [1, 2] 不好吗? 这样最少和下面
//   推入 3, 4, 5 这样每项之间添加逗号的写法是一致的, 现在感觉为强行省略而省略.
push(nums, 3, 4, 5)
```

  - (3.2) `pop(expr)` : 从表达式的末尾推出一个值.

```
nums = 4 5 3 2 1
num = pop(nums)
nums                    // => 4 5 3 2
num                     // => 1
```

  - (3.3) `shift(expr)` : 从表达式的开头推出一个值.

```
nums = 4 5 3 2 1
num = shift(nums)
nums                    // => 5 3 2 1
num                     // => 4
```

  - (3.4) `unshift(expr, args...)` : 将一个或多个值添加到表达式的开头. 又名 `prepend()`

```
nums = 4 5
unshift(nums, 3, 2, 1)
nums                    // => 1 2 3 4 5
```

  - (3.5) `index(list, value)` : 返回列表中要搜索值的索引(从 0 开始).

```
list = 1 2 3
index(list, 2)              // => 1
index(1px solid red, red)   // => 2
```

- (3.6) `keys(pairs)` : 返回给定键值对中的键

```
pairs = (one 1) (two 2) (three 3)
keys(pairs)                 // => one two three
```

- (3.7) `values(pairs)` : 返回给定键值对中的值

```
pairs = (one 1) (two 2) (three 3)
values(pairs)               // => 1 3 43
```

- (3.8) `list-separator(list)`
- (3.9) `length([expr])` : 返回给定表达式的长度
- (3.10) `last(expr)` : 返回给定表达式的最后一个值.

- (4) Unit Functions(单元函数)
  - (4.1) `typeof(node)` : 字符串性质返回 `node` 类型. 别名有 `type-of` 和 `type` ,

```
type(12)            // => 'unit'
typeof(12)          // => 'unit
typeof(#fff)        // => 'rgba'
type-of(#fff)       // => 'rgba'
```

  - (4.2) `unit(val[, type])` : 返回 `unit` 类型的字符串或空字符串, 或者赋予 `type` 值而无需单位转换.

```
unit(10)            // => ''
unit(15in)          // => 'in'
unit(15%, 'px')     // => 15px
unit(15%, px)       // => 15px
```

  - (4.3) `percentage(num)`

- (5) Math Functions(数学函数)
  - (5.1) `abs(unit)` : 绝对值.
  - (5.2) `ceil(unit)` : 向上取整
  - (5.3) `floor(unit)` : 向下取整
  - (5.4) `round(unit)` : 四舍五入取整
  - (5.5) `sin(angle)` :
  - (5.6) `cos(angle)`
  - (5.7) `tan(angle)`
  - (5.8) `min(a, b)` : 取较小值
  - (5.9) `max(a, b)` : 取较大值
  - (5.10) `even(unit)` : 是否为偶数

- (5.11) `odd(unit)` : 是否为奇数
- (5.12) `sum(nums)` : 求和
- (5.13) `avg(nums)` : 求平均数
- (5.14) `range(start, stop[, step])`
- (5.15) `base-convert(num, base, width)`

- (6) String Functions(字符串函数)
  - (6.1) `match(pattern, string[, flags])` : 检测 `string` 是否匹配给定的 `pattern` (模式)

    ```
    match('^foo(bar)?', foo)            // => true
    match('^foo(bar)?', foobar)         // => true
    ```

  - (6.2) `replace(pattern, replacement, val)`
  - (6.3) `join(delim, vals...)` : 把给定的值使用分隔符(delimiter)连接
  - (6.4) `split(delim, val)`
  - (6.5) `substr(val, start, length)`
  - (6.6) `slice(val, start[, end])`
  - (6.7) `unquote(str | ident)` : 去除给定 string 的引号, 返回字面量节点
  - (6.8) `convert(str)`
  - (6.9) `s(fmt, ...)`

- (7) Utility Functions(实用函数)
  - (7.1) `called-from` 属性
  - (7.2) `current-media()`
  - (7.3) `+cache(keys...)`
  - (7.4) `+prefix-classes(prefix)`
  - (7.5) `lookup(name)`
  - (7.6) `define(name, expr[, global])`
  - (7.7) `operate(op, left, right)` : 在左(left)和右(right) 操作对象上执行给定的操作(operation).
  - (7.8) `selector()`
  - (7.9) `selector-exists(selector)`
  - (7.10) `opposite-position(positions)` : 返回给定位置(position) 相反内容.
  - (7.11) `image-size(path)` : 返回指定路径(path) 图片的宽(width)和 高(height). 向上查找路径的方法和 `@import` 一样, `paths` 设置的时候改变.
  - (7.12) `embedurl(path[, encoding])`
  - (7.13) `add-property(name, expr)`

- (8) Console Functions(控制台函数)
  - (8.1) `warn(msg)` : 使用给定的信息, 发出警告, 并不退出.
  - (8.2) `error(msg)` : 使用给定的信息, 发出错误, 并退出.

    - (8.3) `p(expr)` : 检查给定的表达式
- (8) External File Functions(外部文件函数)
    - (9.1) `json(path[, options])`

1.8 Rest parameters (剩余参数 / 其余参数)

- (1) `args...` Stylus 支持 `name...` 形式的剩余参数. 这在处理浏览器私有属性, 如 `-moz` 或 `-webkit` 的时候很有用.

  下面这个例子中, 所有的参数(1px, 2px, ...)都被一个 args 参数给简单消化了:

  ```
  box-shadow(args...)
      -webkit-box-shadow args
      -moz-box-shadow args
      box-shadow args
  #log
      box-shadow 1px 2px 5px #eee
  ```

  生成为:

  ```
  #login {
      -webkit-box-shadow: 1px 2px 5px #eee;
      -moz-box-shadow: 1px 2px 5px #eee;
      box-shadow: 1px 2px 5px #eee;
  }
  ```

  如果我们想指定特定的参数, 如 `x-offset` , 我们可以使用 `args[0]` , 或者, 我们可能希望重新定义混入.

  ```
  box-shadow(offset-x, args...)
      got-offset-x offset-x
      -webkit-box-shadow args
      -moz-box-shadow args
      box-shadow args
  #log
      box-shadow 1px 2px 5px #eee
  ```

  生成为:

  ```
  #login {
      got-offset-x: 1px;
      -webkit-box-shadow: 1px 2px 5px #eee;
      -moz-box-shadow: 1px 2px 5px #eee;
      box-shadow: 1px 2px 5px #eee;
  }
  ```

- (2) `arguments` `arguments` : `arguments` 变量可以实现表达式的精确传递, 包括逗号等等. 这可以弥补 `args...` 参数的一些不足, 见下面的例子:

```
box-shadow(args...)
    -webkit-box-shadow args
    -moz-box-shadow args
    box-shadow args
#login
    box-shadow #ddd 1px 1px, #eee 2px 2px
```

生成为:

```
#login {
    -webkit-box-shadow: #ddd 1px 1px #eee 2px 2px;
    -moz-box-shadow: #ddd 1px 1px #eee 2px 2px;
    box-shadow: #ddd 1px 1px #eee 2px 2px;
}
```

逗号被忽略了, 我们现在使用 `arguments` 重新定义这个混合书写.

```
box-shadow()
    -webkit-box-shadow arguments
    -moz-box-shadow arguments
    box-shadow arguments
#login
    box-shadow #ddd 1px 1px, #eee 2px 2px
```

生成为:

```
#login {
    -webkit-box-shadow: #ddd 1px 1px, #eee 2px 2px;
    -moz-box-shadow: #ddd 1px 1px, #eee 2px 2px;
    box-shadow: #ddd 1px 1px, #eee 2px 2px;
}
```

现在, 结果便是我们想要的了.

1.9 注释 (comment)

Stylus 支持三种注释方式: 单行注释、多行注释和多行缓冲注释.

- (1) 单行注释: 单行注释和 JavaScript 中的注释方式一样, 不会输出到最终的 CSS 中:

```
// I'm a comment!
body
    padding 5px // some awesome padding
```

- (2) 多行注释: 多行注释看起来有点像 CSS 的常规注释. 然而, 它们只有在 `compress` 选项未被启用的时候才会被输出.

```
/*
 * Adds the given numbers together
 */
 add(a, b)
    return a + b
```

- (3) 多行缓冲注释: 以 `/*!` 开头的多行注释将被输出. 这其实是告诉 stylus 无论是否压缩(开启 `compress` 选项) 都直接输出此注释.

```
/*!
 * Adds the given numbers together
 */
 add(a, b)
    return a + b
```

1.10 Conditionals (条件句)

条件语句提供对静态语言的控制流, 并提供条件导入(), mixin, 函数等。以下示例仅是示例, 不建议使用:)

如果/否则, 如果/否则 if条件可以按您期望的那样工作, 只需接受一个表达式, 则在为true时评估以下块。典型的else, if和else标记与if一起用作备用。

1.11 Hashes (哈希)

- (1) 定义: 可以使用大括号来定义哈希对象并用冒号来区分键和值:

```
foo = {
    bar: baz,
    'baz': raz,
    '0': rrr
}
```

使用 "方括号" 和 字符串 来为其赋值:

```
foo = {}
foo['bar'] = baz
foo['baz'] = raz
```

请注意, 然不能在花括号定义中使用变量和插值, 但可以在方括号内使用变量:

```
foo = {}
bar = 'baz'
foo[baz] = raz

foo.baz     // => raz
```

- (2) 取值: 和 js 中对象的取值方式相同, 有 2 种方式: 点表示法 和 方括号表示法.

```
foo = {bar: 'baz'}
foo.bar     // => "baz"
foo['bar']  // => "baz"
```

1.12 Iteration (迭代)

1.13 `@import` and `@require`

- Stylus支持字面 `@import` CSS, 也支持其他 Stylus 样式的动态导入.

```
@import "reset.css"

@import "~assets/stylus/variable";
```

1.14 `@media`

1.15 `@font-face`

1.16 `@keyframes`

1.17 Other `@-Rules`

1.18 `@extend`

1.19 `@block`

1.20 `url()`

1.21 CSS Literal (CSS 字面量)

- 不管什么原因, 如果遇到 Stylus 搞不定的特殊需求, 你都可以使用 `@css` 直接书写普通的 CSS 代码:

```
@css {
    .ie-opacity {
        filter: progid:DXImageTransform.Microsoft.Alpha(opacity=25);
        -ms-filter: "progid:DXImageTransform.Microsoft.Alpha(opacity=25)";
    }
}
```

编译为:

```
.ie-opacity {
    filter: progid:DXImageTransform.Microsoft.Alpha(opacity=25);
    -ms-filter: "progid:DXImageTransform.Microsoft.Alpha(opacity=25)";
}
```

1.22 CSS Style Syntax (CSS 样式解析)

1.23 Char Escaping (字符转码)

- Stylus 能让你对字符转码. 这可以将字符变为标识符(identifiers), 继而输出为字面量. 例如:

```
body
    padding 1 \+ 2
```

编译为:

```
body {
    padding: 1 + 2;
}
```

注意, 当用作属性时, stylus 需要将 `/` 用括号括起来:

```
body
    font 14px/1.4
    font (14px/1.4)
```

编译为:

```
body {
    font: 14px/1.4;
    font: 10px;
}
```

1.24 Executable (可执行的)

1.25 Error Reporting (错误报告)

1.26 Connect middleware (连接中间件)

1.27 Introspection API (自检 API)

1.28 JavaScript API

1.29 SourceMaps (资源图)

1.30 CSS3 extensions with NIB (Nib 下 CSS 扩展)

2. 在 Vue 组件中使用 stylus

- 比如在 Vue 项目的 `src/assets` 文件夹下有个 `stylus` 的文件夹, 里面有这几个文件:
  - `base.styl`
  - `icon.styl`
  - `index.styl`
  - `mixin.styl`
  - `reset.styl`
  - `variable.styl`

(2.1) 在 Vue 的单文件组件中引入 stylus

```
<template>
    <div>
    </div>
</template>
<script>
</script>
<style scoped lang="stylus" rel="stylesheet/stylus">
    @import "~assets/stylus/variable";
</style>
```

(2.2) 定义变量

```
// - variable.styl

// - 颜色定义规范
$color-background-black = #222
$color-highlight-background = #333
$color-dialog-background = #666
$color-theme = #ffcd32
$color-theme-d = rgba(255, 205, 49, .5)
$color-sub-theme = #d93f30
$color-text = #fff
$color-text-d = rgba(255, 255, 255, .3)
$color-text-l = rgba(255, 255, 255, .5)
$color-text-ll = rgba(255, 255, 255, .8)

// - 字体定义规范
$font-size-medium = 13px
$font-size-medium-x = 15px
$font-size-large = 18px
$font-size-large-x = 22px
```