3d-tiles Styling

Contents(内容)

Overview（概述）

3D Tiles styles提供了tileset features的简介声明样式. 样式定义表达式来评估feature的显示, 例如`color` (RGB和透明) 以及 `show` 属性, 通常基于存储在tile的批处理表（Batch Table）中的特征属性.
3D Tiles styles provide concise declarative styling of tileset features. A style defines expressions to evaluate the display of a feature, for example color (RGB and translucency) and show properties, often based on the feature's properties stored in the tile's Batch Table.

样式可以应用于不包含features的tile，在这种情况下，tile被视为没有属性的单个feature。
A style may be applied to a tile that doesn't contain features, in which case the tile is treated as an implicit single feature without properties.

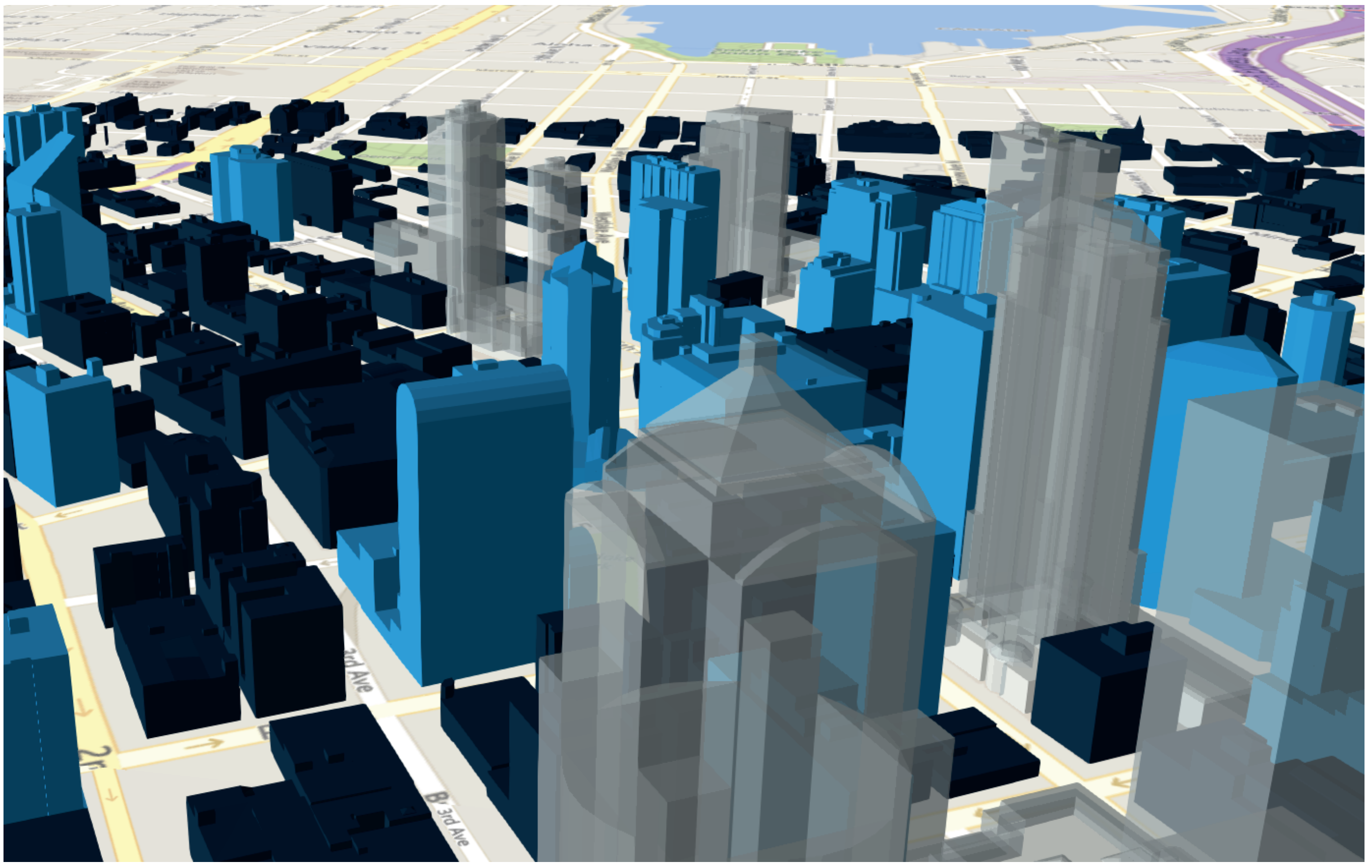
虽然可以为tileset的属性创建和引用样式，但样式独立于tileset，因此任何样式都可以应用于任何tileset。
While a style may be created for and reference properties of a tileset, a style is independent of a tileset, such that any style can be applied to any tileset.

https://github.com/ElevenIjusee

样式是用 JSON 定义的，表达式是用 JavaScript 的一个小子集编写的，用于样式增强。此外，样式语言提供了一组内置函数来支持常见的数学运算。
Styles are defined with JSON and expressions written in a small subset of JavaScript augmented for styling. Additionally, the styling language provides a set of built-in functions to support common math operations.

以下示例根据建筑物高度分配颜色。
The following example assigns a color based on building height.

```
{
    "show" : "${Area} > 0",
    "color" : {
        "conditions" : [
            ["${Height} < 60", "color('#13293D')"],
            ["${Height} < 120", "color('#1B98E0')"],
            ["true", "color('#E8F1F2', 0.5)"]
        ]
    }
}
```



Concepts（概念）

Styling features（样式要素（特征））

可用于样式要素的可见性是`show`属性，其分配的表达式将计算成一个布尔值，以确定该feature是否可见。`color`属性，其分配的表达式将计算为一个Color对象（RGB 和半透明），该对象确定feature的显示颜色。
Visual properties available for styling features are the `show` property, the assigned expression of which will

evaluate to a boolean that determines if the feature is visible, and the `color` property, the assigned expression of which will evaluate to a `Color` object (RGB and translucency) which determines the displayed color of a feature.

以下样式为每个feature分配默认显示和颜色属性：
The following style assigns the default show and color properties to each feature:

```
{
    "show" : "true",
    "color" : "color('#ffffff')"
}
```

`show`属性可以通过表达式设置只显示特定的feature，而不是全部的features。例如，以下表达式将仅显示ZipCode为19341的feature
Instead of showing all features, `show` can be an expression dependent on a feature's properties, for example, the following expression will show only features in the 19341 zip code:

```
{
    "show" : "${ZipCode} === '19341'"
}
```

`show`属性也可以用于更复杂的查询，例如，这里使用复合条件跟正则表达式仅显示县市的`Chest`属性以1970年开始建造并且建造年份大于或等于1970的feature。
`show` can also be used for more complex queries; for example, here a compound condition and regular expression are used to show only features whose county starts with `'Chest'` and whose year built is greater than or equal to 1970:

```
{
    "show" : "(regExp('^Chest').test(${County})) && (${YearBuilt} >= 1970)"
}
```

颜色也可以由依赖于feature的属性的表达式定义。例如，以下颜色表达式代表温度高于90的为红色，其他则为白色。
Colors can also be defined by expressions dependent on a feature's properties. For example, the following expression colors features with a temperature above 90 as red and the others as white:

```
{
    "color" : "(${Temperature} > 90) ? color('red') : color('white')"
}
```

颜色的alpha分量定义了feature的不透明度。例如以下内容根据feature的属性设置feature的RGB颜色分量，并使体积大于100的feature透明。
The color's alpha component defines the feature's opacity. For example, the following sets the feature's RGB

https://github.com/ElevenIjusee

color components from the feature's properties and makes features with volume greater than 100 transparent:

```
{
    "color" : "rgba(${red}, ${green}, ${blue}, (${volume} > 100 ? 0.5 : 1.0))"
}
```

Conditions（条件）

除了包含表达式的字符串，`color`跟`show`还可以是定义一系列条件的数组（类似于if...else语句）。例如，可以用于制作任何类型的包含/不包含间隔的颜色图，颜色渐变 In addition to a string containing an expression, `color` and `show` can be an array defining a series of conditions (similar to `if...else` statements). Conditions can, for example, be used to make color maps and color ramps with any type of inclusive/exclusive intervals.

例如，以下表达式将ID属性与颜色进行映射。conditions按照顺序计算，因此如果`${id}`不是1或2，则`true`条件返回白色。如果不满足任何条件，则feature的颜色将为`undefined`；
For example, the following expression maps an ID property to colors. Conditions are evaluated in order, so if `${id}` is not `'1'` or `'2'`, the `"true"` condition returns white. If no conditions are met, the color of the feature will be `undefined`:

```
{
    "color" : {
        "conditions" : [
            ["${id} === '1'", "color('#FF0000')"],
            ["${id} === '2'", "color('#00FF00')"],
            ["true", "color('#FFFFFF')"]
        ]
    }
}
```

下一个示例说明如何使用condition创建颜色渐变，包含下边界（下限）但不包含上边界（上限）。
The next example shows how to use conditions to create a color ramp using intervals with an inclusive lower bound and exclusive upper bound:

```
"color" : {
    "conditions" : [
        ["(${Height} >= 1.0)  && (${Height} < 10.0)", "color('#FF00FF')"],
        ["(${Height} >= 10.0) && (${Height} < 30.0)", "color('#FF0000')"],
        ["(${Height} >= 30.0) && (${Height} < 50.0)", "color('#FFFF00')"],
        ["(${Height} >= 50.0) && (${Height} < 70.0)", "color('#00FF00')"],
        ["(${Height} >= 70.0) && (${Height} < 100.0)", "color('#00FFFF')"],
        ["(${Height} >= 100.0)", "color('#0000FF')"]
    ]
}
```

由于条件是按顺序计算的，所以上面可以更简洁地写成如下：
Since conditions are evaluated in order, the above can be written more concisely as the following:

```
"color" : {
    "conditions" : [
        ["(${Height} >= 100.0)", "color('#0000FF')"],
        ["(${Height} >= 70.0)", "color('#00FFFF')"],
        ["(${Height} >= 50.0)", "color('#00FF00')"],
        ["(${Height} >= 30.0)", "color('#FFFF00')"],
        ["(${Height} >= 10.0)", "color('#FF0000')"],
        ["(${Height} >= 1.0)", "color('#FF00FF')"]
    ]
}
```

Defining variables（定义变量）

常用的表达式可以以变量名作为key储存在`defines`对象中。如果变量引用已定义表达式的名称，则将其替换为引用的计算表达式的结果；
Commonly used expressions may be stored in a `defines` object with a variable name as a key. If a variable references the name of a defined expression, it is replaced with the result of the referenced evaluated expression:

```
{
    "defines" : {
        "NewHeight" : "clamp((${Height} - 0.5) / 2.0, 1.0, 255.0)",
        "HeightColor" : "rgb(${Height}, ${Height}, ${Height})"
    },
    "color" : {
        "conditions" : [
            ["(${NewHeight} >= 100.0)", "color('#0000FF') * ${HeightColor}"],
            ["(${NewHeight} >= 50.0)", "color('#00FF00') * ${HeightColor}"],
            ["(${NewHeight} >= 1.0)", "color('#FF0000') * ${HeightColor}"]
        ]
    },
    "show" : "${NewHeight} < 200.0"
}
```

定义表达式不能引用其他定义，但是，可以引用同名的feature属性。在下面的样式中，高度为150的feature的颜色为红色。
A define expression may not reference other defines; however, it may reference feature properties with the same name. In the style below a feature of height 150 gets the color red:

```
{
    "defines" : {
        "Height" : "${Height}/2.0}",
    },
    "color" : {
        "conditions" : [
            ["(${Height} >= 100.0)", "color('#0000FF')"],
            ["(${Height} >= 1.0)", "color('#FF0000')"]
        ]
    }
}
```

Meta property（元属性）

可以使用`meta`属性来定义feature的非视觉属性。例如，以下将description元属性设置为包含功能名称的字符串：
Non-visual properties of a feature can be defined using the `meta` property. For example, the following sets a `description` meta property to a string containing the feature name:

```
{
    "meta" : {
        "description" : "'Hello, ${featureName}.'"
    }
}
```

元属性表达式可以计算为任何类型，例如：
A meta property expression can evaluate to any type. For example:

```
{
    "meta" : {
        "featureColor" : "rgb(${red}, ${green}, ${blue})",
        "featureVolume" : "${height} * ${width} * ${depth}"
    }
}
```

Expressions（表达式）

表达式使用的语言是JavaScript (EMCAScript 5)的一个小子集，加上原生向量和正则表达式类型以及只读变量的形式访问tileset的feature属性。
The language for expressions is a small subset of JavaScript (EMCAScript 5), plus native vector and regular expression types and access to tileset feature properties in the form of readonly variables.

> **Implementation Note:** CesiumJS uses the jsep JavaScript expression parser library to parse style expressions into an abstract syntax tree (AST).

Semantics（语义）

点符号可以用于按名称访问属性，例如，`building.name`。
Dot notation is used to access properties by name, e.g., `building.name`.

方括号(`[]`)也可以用于访问属性，例如`building['name']`, 或者访问数组, 例如, `temperatures[1]`.
Bracket notation (`[]`) is also used to access properties, e.g., `building['name']`, or arrays, e.g., `temperatures[1]`.

使用括号 (`()`)和逗号分隔的参数调用函数，例如(`isNaN(0.0)`, `color('cyan', 0.5)`)。
Functions are called with parenthesis (`()`) and comma-separated arguments, e.g., (`isNaN(0.0)`, `color('cyan', 0.5)`).

Operators（运算符）

支持以下运算符，其语义和优先级与JavaScript相同。
The following operators are supported with the same semantics and precedence as JavaScript.

- Unary（一元）: `+`, `-`, `!`
  - Not supported: `~`
- Binary（二元）: `||`, `&&`, `===`, `!==`, `<`, `>`, `<=`, `>=`, `+`, `-`, `*`, `/`, `%`, `=~`, `!~`
  - Not supported（不支持）: `|`, `^`, `&`, `<<`, `>>`, and `>>>`
- Ternary（三元）: `? :`

并且还支持对表达式进行分组以提高清晰度和优先级。
`(` and `)` are also supported for grouping expressions for clarity and precedence.

逻辑`||` 和 `&&`实现短路； `true || expression`不会计算右边的表达式，`false && expression`也不会计算右边的表达式
Logical `||` and `&&` implement short-circuiting; `true || expression` does not evaluate the right expression, and `false && expression` does not evaluate the right expression.

同理，`true ? leftExpression : rightExpression`只执行左边的表达式，`false ? leftExpression : rightExpression`只执行右边的表达式。
Similarly, `true ? leftExpression : rightExpression` only executes the left expression, and `false ? leftExpression : rightExpression` only executes the right expression.

Types（类型）

The following types are supported:

- `Boolean`
- `Null`
- `Undefined`
- `Number`
- `String`
- `Array`
- `vec2`
- `vec3`
- `vec4`
- `RegExp`

除了`vec2`, `vec3`, `vec4`和`RegExp`之外的所有类型都具有与JavaScript相同的语法和运行时的行为。`vec2`, `vec3` 和 `vec4`是派生自GLSL向量，其行为类似于JavaScript的`Object` (请参阅Vector section（向量章节）)。Colors源自CSS3 Colors并且实现为`vec4`。`RegExp`源自JavaScript并且在RegExp section（正则章节）进行了描述。

https://github.com/ElevenIjusee

All of the types except `vec2`, `vec3`, `vec4`, and `RegExp` have the same syntax and runtime behavior as JavaScript. `vec2`, `vec3`, and `vec4` are derived from GLSL vectors and behave similarly to JavaScript `Object` (see the Vector section). Colors derive from CSS3 Colors and are implemented as `vec4`. `RegExp` is derived from JavaScript and described in the RegExp section.

不同类型的表达式示例如下；
Example expressions for different types include the following:

- `true`, `false`
- `null`
- `undefined`
- `1.0`, `NaN`, `Infinity`
- `'Cesium'`, `"Cesium"`
- `[0, 1, 2]`
- `vec2(1.0, 2.0)`
- `vec3(1.0, 2.0, 3.0)`
- `vec4(1.0, 2.0, 3.0, 4.0)`
- `color('#00FFFF')`
- `regExp('^Chest'))`

Number（数字类型）

在JavaScript中数字可以是`NaN`或`Infinity`。以下的测试函数是可以被支持的；
As in JavaScript, numbers can be NaN or Infinity. The following test functions are supported:

- `isNaN(testValue : Number) : Boolean`
- `isFinite(testValue : Number) : Boolean`

String（字符串类型）

字符串以UTF-8编码。
Strings are encoded in UTF-8.

Vector（向量/矢量）

style语言包括2，3, 4组分量的浮点向量类型；`vec2`, `vec3`和`vec4`。向量构造函数与GLSL共享相同的规则。
The styling language includes 2, 3, and 4 component floating-point vector types: `vec2`, `vec3`, and `vec4`. Vector constructors share the same rules as GLSL:

vec2

- `vec2(xy : Number)` - 用一个数字初始化每个组件。initialize each component with the number
- `vec2(x : Number, y : Number)` - 用两个数字初始化。initialize with two numbers
- `vec2(xy : vec2)` - 用另一个`vec2`初始化。initialize with another `vec2`
- `vec2(xyz : vec3)` - 使用`vec3`前两个分量，舍弃第三个分量。drops the third component of a `vec3`
- `vec2(xyzw : vec4)` - 使用`vec4`前两个分量，舍弃第三，第四个分量。drops the third and fourth component of a `vec4`

vec3

- `vec3(xyz : Number)` - 用一个数字初始化每个组件。initialize each component with the number
- `vec3(x : Number, y : Number, z : Number)` - 用三个数字初始化。initialize with three numbers
- `vec3(xyz : vec3)` - 通过另一个`vec3`来初始化。initialize with another `vec3`
- `vec3(xyzw : vec4)` - 通过`vec4`来初始化，舍弃第四个分量。drops the fourth component of a `vec4`
- `vec3(xy : vec2, z : Number)` - 通过一个`vec2`和一个数字来初始化。initialize with a `vec2` and number
- `vec3(x : Number, yz : vec2)` - 同上，但顺序不同。initialize with a `vec2` and number

vec4

- `vec4(xyzw : Number)` - 通过一个数字来初始化。initialize each component with the number
- `vec4(x : Number, y : Number, z : Number, w : Number)` - 通过四个数字来初始化。initialize with four numbers
- `vec4(xyzw : vec4)` - 通过另一个`vec4`向量来初始化。initialize with another `vec4`
- `vec4(xy : vec2, z : Number, w : Number)` - 通过一个`vec2`和两个数字来初始化。initialize with a `vec2` and two numbers
- `vec4(x : Number, yz : vec2, w : Number)` - 同上。initialize with a `vec2` and two numbers
- `vec4(x : Number, y : Number, zw : vec2)` - 同上。initialize with a `vec2` and two numbers
- `vec4(xyz : vec3, w : Number)` - 通过一个`vec3`跟一个数字来初始化。initialize with a `vec3` and number
- `vec4(x : Number, yzw : vec3)` - 同上。initialize with a `vec3` and number

Vector usage（向量用法）

`vec2` 可以通过以下方式访问分量。components may be accessed with

- `.x`, `.y`
- `.r`, `.g`
- `[0]`, `[1]`

`vec3` 可以通过以下方式访问分量。components may be accessed with

- `.x`, `.y`, `.z`
- `.r`, `.g`, `.b`
- `[0]`, `[1]`, `[2]`

`vec4` 可以通过以下方式访问分量。components may be accessed with

- `.x`, `.y`, `.z`, `.w`
- `.r`, `.g`, `.b`, `.a`
- `[0]`, `[1]`, `[2]`, `[3]`

与GLSL不同，style语言不支持swizzling。例如，`vec3(1.0).xy`是不被支持的。
Unlike GLSL, the styling language does not support swizzling. For example, `vec3(1.0).xy` is not supported.

向量支持以下一元运算符：`-`, `+`。
Vectors support the following unary operators: `-`, `+`.

向量通过逐个分量操作支持下列二元运算符：`===`, `!==`, `+`, `-`, `*`, `/`, and `%`。例如 `vec4(1.0) === vec4(1.0)`为真，是因为它们 *x*, *y*, *z*, 和 *w* 分量都相等。对于`vec2`, `vec3`和`vec4`运算符本质上是重载的。

Vectors support the following binary operators by performing component-wise operations: `===`, `!==`, `+`, `-`, `*`, `/`, and `%`. For example `vec4(1.0) === vec4(1.0)` is true since the *x*, *y*, *z*, and *w* components are equal. Operators are essentially overloaded for `vec2`, `vec3`, and `vec4`.

`vec2`,`vec3`和`vec4`具有一个`toString`函数去显式（和隐式）地转换为字符串。转换后格式为`'(x, y)'`,`'(x, y, z)'`和`'(x, y, z, w)'`。
`vec2`, `vec3`, and `vec4` have a `toString` function for explicit (and implicit) conversion to strings in the format `'(x, y)'`, `'(x, y, z)'`, and `'(x, y, z, w)'`.

- `toString() : String`

`vec2`,`vec3`和`vec4`不暴露任何其他函数或者prototype对象。
`vec2`, `vec3`, and `vec4` do not expose any other functions or a `prototype` object.

Color（颜色）

颜色由`vec4`实现，通过下列方式之一创建：
Colors are implemented as `vec4` and are created with one of the following functions:

- `color()`
- `color(keyword : String, [alpha : Number])`
- `color(6-digit-hex : String, [alpha : Number])`
- `color(3-digit-hex : String, [alpha : Number])`
- `rgb(red : Number, green : Number, blue : Number)`
- `rgba(red : Number, green : Number, blue : Number, alpha : Number)`
- `hsl(hue : Number, saturation : Number, lightness : Number)`
- `hsla(hue : Number, saturation : Number, lightness : Number, alpha : Number)`
  不传参数调用`color()`等效于调用 `color('#FFFFFF')`。
  Calling `color()` with no arguments is the same as calling `color('#FFFFFF')`.

通过不区分大小写的关机键字（如`'cyan'`）或十六进制rgb定义的颜色作为字符串传递给`color`函数。例如：
Colors defined by a case-insensitive keyword (e.g., `'cyan'`) or hex rgb are passed as strings to the `color` function. For example:

- `color('cyan')`
- `color('#00FFFF')`
- `color('#0FF')`

这些`color`函数有一个可选的第二参数，它是定义不透明度的分量。`0.0`完全透明，`1.0`完全不透明。例如：
These `color` functions have an optional second argument that is an alpha component to define opacity, where `0.0` is fully transparent and `1.0` is fully opaque. For example:

- `color('cyan', 0.5)`

十进制RGB或者HSL定义的颜色需要分别通过`rgb`和`hsl`函数创建，就像在CSS中一样（但是百分比范围是从`0.0`到`1.0`代表从`0%`到`100%`），例如；
Colors defined with decimal RGB or HSL are created with `rgb` and `hsl` functions, respectively, just as in CSS (but with percentage ranges from `0.0` to `1.0` for `0%` to `100%`, respectively). For example:

- `rgb(100, 255, 190)`

- `hsl(1.0, 0.6, 0.7)`

`rgb`分量的范围是从`0`到`255`。对于`hsl`，色相、饱和度和亮度的范围是从`0.0`到`1.0`。
The range for `rgb` components is `0` to `255`, inclusive. For `hsl`, the range for hue, saturation, and lightness is `0.0` to `1.0`, inclusive.

通过`rgba`或者`hsla`定义的颜色有第四个参数，它是用来定义不透明度的alpha分量。`0.0`完全透明，`1.0`完全不透明。例如：
Colors defined with `rgba` or `hsla` have a fourth argument that is an alpha component to define opacity, where `0.0` is fully transparent and `1.0` is fully opaque. For example:

- `rgba(100, 255, 190, 0.25)`
- `hsla(1.0, 0.6, 0.7, 0.75)`

颜色等同于`vec4`类型并且共享相同的方法，运算符和分量访问方式。颜色的储存范围为`0.0`到`1.0`。
Colors are equivalent to the `vec4` type and share the same functions, operators, and component accessors. Color components are stored in the range `0.0` to `1.0`.

例如:

- `color('red').x`, `color('red').r` 和 `color('red')[0]` 都为 `1.0`.
- `color('red').toString()` 得出 `(1.0, 0.0, 0.0, 1.0)`
- `color('red') * vec4(0.5)` 得出 `vec4(0.5, 0.0, 0.0, 0.5)`

RegExp（正则表达式）

正则表达式使用以下函数创建，其行为类似于JavaScript的`RegExp`构造函数。
Regular expressions are created with the following functions, which behave like the JavaScript `RegExp` constructor:

- `regExp()`
- `regExp(pattern : String, [flags : String])`

不传参数使用`regExp()`等同于调用`regExp('(?:)')`。
Calling `regExp()` with no arguments is the same as calling `regExp('(?:)')`.

如果给定参数，`flags`可以是具有以下值的任意组合；
If specified, `flags` can have any combination of the following values:

- `g` - global match
- `i` - ignore case
- `m` - multiline
- `u` - unicode
- `y`- sticky

正则表达式支持以下函数。
Regular expressions support these functions:

- `test(string : String) : Boolean` - 测试指定的字符串是否匹配。Tests the specified string for a match.

- `exec(string : String) : String` - 在指定的字符串中搜索匹配项，如果成功则返回已捕获的第一个`String`实例。如果失败则返回`null`。Executes a search for a match in the specified string. If the search succeeds, it returns the first instance of a captured `String`. If the search fails, it returns `null`.

For example:

```
{
    "Name" : "Building 1"
}
```

```
regExp('a').test('abc') === true
regExp('a(.)', 'i').exec('Abc') === 'b'
regExp('Building\s(\d)').exec(${Name}) === '1'
```

正则表达式具有`toString`函数，可以显式（和隐式）转换为格式为`'pattern'`的字符串。
Regular expressions have a `toString` function for explicit (and implicit) conversion to strings in the format `'pattern'`:

- `toString() : String`

正则表达式不暴露其他任何函数方法或`prototype`对象。
Regular expressions do not expose any other functions or a `prototype` object.

运算符`=~`和`!~`为正则表达式重载。`=~`运算符的行为跟`test`函数相匹配，并且测试指定的字符串是否匹配。如果找到则返回`true`，如果没有找到则返回`false`。`!~`运算符是`=~`的逆操作。如果找到则返回`true`，如果没有找到则返回`false`。这两个运算符都是可以交换的（符号两边交换结果不变）。
The operators `=~` and `!~` are overloaded for regular expressions. The `=~` operator matches the behavior of the `test` function, and tests the specified string for a match. It returns `true` if one is found, and `false` if not found. The `!~` operator is the inverse of the `=~` operator. It returns `true` if no matches are found, and `false` if a match is found. Both operators are commutative.

例如，以下表达式的计算结果都为真。
For example, the following expressions all evaluate to true:

```
regExp('a') =~ 'abc'
'abc' =~ regExp('a')

regExp('a') !~ 'bcd'
'bcd' !~ regExp('a')
```

Operator rules（运算规则）

- 一元运算符`+`和`-` 仅可以对数字和向量表达式使用。
- Unary operators `+` and `-` operate only on number and vector expressions.
- 一元运算符`!`仅可以对布尔表达式使用。

- Unary operator `!` operates only on boolean expressions.
- 二元运算符`<`, `<=`, `>`和`>=`仅可以对数字表达式使用。
- Binary operators `<`, `<=`, `>`, and `>=` operate only on number expressions.
- 二元运算符`||`和`&&`仅可以对布尔表达式使用。
- Binary operators `||` and `&&` operate only on boolean expressions.
- 二元运算符`+`可以对下列表达式使用;
- Binary operator `+` operates on the following expressions:
  - 数字表达式
  - Number expressions
  - 相同类型的向量表达式
  - Vector expressions of the same type
  - 如果至少一个表达式是字符串，则另一个表达式在String Conversions只会也会被转换为字符串，并返回连接的字符串。例如，`"name" + 10`得到`"name10"`。
  - If at least one expressions is a string, the other expression is converted to a string following String Conversions, and the operation returns a concatenated string, e.g. `"name" + 10` evaluates to `"name10"`
- 二元运算符`-`对下列表达式生效:
- Binary operator `-` operates on the following expressions:
  - 数字表达式
  - Number expressions
  - 相同类型的向量表达式
  - Vector expressions of the same type
- 二元运算符`*`对下列表达式生效:
- Binary operator `*` operates on the following expressions:
  - 数字表达式
  - Number expressions
  - 相同类型的向量表达式
  - Vector expressions of the same type
  - 数字表达式跟向量表达式的混合表达式。例如`3 * vec3(1.0)`和`vec2(1.0) * 3`
  - Mix of number expression and vector expression, e.g. `3 * vec3(1.0)` and `vec2(1.0) * 3`
- 二元运算符`/`对下列表达式生效;
- Binary operator `/` operates on the following expressions:
  - 数字表达式
  - Number expressions
  - 相同类型的向量表达式
  - Vector expressions of the same type
  - 向量表达式后面跟着数字表达式。例如`vec3(1.0) / 3`
  - Vector expression followed by number expression, e.g.`vec3(1.0) / 3`
- 二元运算符`%`对下列表达式生效;
- Binary operator `%` operates on the following expressions:
  - 数字表达式
  - Number expressions
  - 相同类型的向量表达式
  - Vector expressions of the same type
- 二元等于运算符`===`和`!==`对任何表达式都生效。如果两边表达式类型不匹配则返回 `false`

- Binary equality operators `===` and `!==` operate on any expressions. The operation returns `false` if the expression types do not match.
- 二元正则表达式`=~`和`!~`的参数需要一个字符串作和一个`RegExp`正则表达式
- Binary `regExp` operators `=~` and `!~` require one argument to be a string expression and the other to be a `RegExp` expression.
- 三元运算符`? :`条件参数必须是布尔表达式
- Ternary operator `? :` conditional argument must be a boolean expression.

Type conversions（类型转换）

基本类型间的显式转换用`Boolean`，`Number`和`String`函数方法处理。
Explicit conversions between primitive types are handled with `Boolean`, `Number`, and `String` functions.

- `Boolean(value : Any) : Boolean`
- `Number(value : Any) : Number`
- `String(value : Any) : String`

For example:

```
Boolean(1) === true
Number('1') === 1
String(1) === '1'
```

`Boolean`和`Number`遵循JavaScript规定。`String`遵循String Conversions。
`Boolean` and `Number` follow JavaScript conventions. `String` follows String Conversions.

这些本质上是强制转换，而不是构造函数。
These are essentially casts, not constructor functions.

除非上面有说明，否则style语言不允许隐式类型转换。例如，表达式`vec3(1.0) === vec4(1.0)`和`"5" < 6`是无效的。
The styling language does not allow for implicit type conversions, unless stated above. Expressions like `vec3(1.0) === vec4(1.0)` and `"5" < 6` are not valid.

String conversions（字符串转换）

`vec2`, `vec3`, `vec4`和`RegExp`表达式使用它们自己的`toString`方法。所有其他类型都遵循JavaScript的约定。
`vec2`, `vec3`, `vec4`, and `RegExp` expressions are converted to strings using their `toString` methods. All other types follow JavaScript conventions.

- `true` - `"true"`
- `false` - `"false"`
- `null` - `"null"`
- `undefined` - `"undefined"`
- `5.0` - `"5"`
- `NaN` - `"NaN"`
- `Infinity` - `"Infinity"`
- `"name"` - `"name"`

- `[0, 1, 2]` - `"[0, 1, 2]"`
- `vec2(1, 2)` - `"(1, 2)"`
- `vec3(1, 2, 3)` - `"(1, 2, 3)"`
- `vec4(1, 2, 3, 4)` - `"(1, 2, 3, 4)"`
- `RegExp('a')` - `"/a/"`

Constants（常量）

style语言支持以下常量：
The following constants are supported by the styling language:

- `Math.PI`
- `Math.E`

PI

数学中的常量PI，表示圆的周长除以其直径，大约为`3.14159`。
The mathematical constant PI, which represents a circle's circumference divided by its diameter, approximately `3.14159`.

```
{
    "show" : "cos(${Angle} + Math.PI) < 0"
}
```

E

欧拉常数和自然对数的底数，大约为`2.71828`。
Euler's constant and the base of the natural logarithm, approximately `2.71828`.

```
{
    "color" : "color() * pow(Math.E / 2.0, ${Temperature})"
}
```

Variables（变量）

变量用于检索tileset的各个feature的属性值。变量使用ES6(ECMAScript 2015)字符串模板语法进行标识，即`${feature.identifier}` 或 `${feature['identifier']}`,其中标识符是区分大小写的属性名称。变量名称以UTF-8 编码。`feature`是隐式的，在大多数情况下可以省略。如果标识符包含非字母非数字字符，例如`:`, `-`, `#`或者空格，应当使用`${feature['identifier']}`这种形式。
Variables are used to retrieve the property values of individual features in a tileset. Variables are identified using the ES 6 (ECMAScript 2015) template literal syntax, i.e., `${feature.identifier}` or `${feature['identifier']}`, where the identifier is the case-sensitive property name. Variable names are encoded in UTF-8. `feature` is implicit and can be omitted in most cases. If the identifier contains non-alphanumeric characters, such as `:`, `-`, `#`, or spaces, the `${feature['identifier']}` form should be used.

变量可以在任何接受有效表达式的地方使用，除了在其他变量标识符内部。例如，下列情况是不允许的；
Variables can be used anywhere a valid expression is accepted, except inside other variable identifiers. For example, the following is not allowed:

```
${foo[${bar}]}
```

如果feature上没有指定的属性名称，则该变量的计算结果为 `undefined`。请注意，如果为某个属性显式存储了“null”，该属性也可以是“null”。
If a feature does not have a property with the specified name, the variable evaluates to `undefined`. Note that the property may also be `null` if `null` was explicitly stored for a property.

变量可以是任何受支持的原生 JavaScript 类型：
Variables may be any of the supported native JavaScript types:

- `Boolean`
- `Null`
- `Undefined`
- `Number`
- `String`
- `Array`

For example:

```
{
    "enabled" : true,
    "description" : null,
    "order" : 1,
    "name" : "Feature name"
}
```

```
${enabled} === true
${description} === null
${order} === 1
${name} === 'Feature name'
```

此外，储存在批处理表Batch Table binary二进制文件中的来自向量属性的变量被视为向量类型：
Additionally, variables originating from vector properties stored in the Batch Table binary are treated as vector types:

`componentType`	**variable type**
`"VEC2"`	`vec2`
`"VEC3"`	`vec3`
`"VEC4"`	`vec4`

变量可用于构造颜色或向量。例如：
Variables can be used to construct colors or vectors. For example:

```
rgba(${red}, ${green}, ${blue}, ${alpha})
vec4(${temperature})
```

点或括号表示法用于访问feature的子属性。例如：
Dot or bracket notation is used to access feature subproperties. For example:

```
{
    "address" : {
        "street" : "Example street",
        "city" : "Example city"
    }
}
```

```
${address.street} === `Example street`
${address['street']} === `Example street`

${address.city} === `Example city`
${address['city']} === `Example city`
```

括号表示法仅支持文字字符串。
Bracket notation supports only string literals.

通过显式使用feature关键字，可以使用括号表示法访问顶级属性。例如：
Top-level properties can be accessed with bracket notation by explicitly using the `feature` keyword. For example:

```
{
    "address.street" : "Maple Street",
    "address" : {
        "street" : "Oak Street"
    }
}
```

```
${address.street} === `Oak Street`
${feature.address.street} === `Oak Street`
${feature['address'].street} === `Oak Street`
${feature['address.street']} === `Maple Street`
```

要访问名叫`feature`的feature，请使用`${feature}`，这相当于调用`${feature.feature}`。
To access a feature named `feature`, use the variable `${feature}`. This is equivalent to accessing `${feature.feature}`

```
{
    "feature" : "building"
}
```

```
${feature} === `building`
${feature.feature} === `building`
```

变量也可以在反引号定义的字符串中替换，例如：
Variables can also be substituted inside strings defined with backticks, for example:

```
{
    "order" : 1,
    "name" : "Feature name"
}
```

```
`Name is ${name}, order is ${order}`
```

括号表示法用于访问feature子属性或者数组。例如：
Bracket notation is used to access feature subproperties or arrays. For example:

```
{
    "temperatures" : {
        "scale" : "fahrenheit",
        "values" : [70, 80, 90]
    }
}
```

```
${temperatures['scale']} === 'fahrenheit'
${temperatures.values[0]} === 70
${temperatures['values'][0]} === 70 // Same as (temperatures[values])[0] and
temperatures.values[0]
```

Built-in functions（内置函数）

style语言支持下列内置函数；
The following built-in functions are supported by the styling language:

- `abs`
- `sqrt`
- `cos`
- `sin`
- `tan`
- `acos`
- `asin`
- `atan`
- `atan2`
- `radians`
- `degrees`
- `sign`
- `floor`
- `ceil`
- `round`
- `exp`
- `log`
- `exp2`
- `log2`
- `fract`
- `pow`
- `min`
- `max`
- `clamp`
- `mix`
- `length`
- `distance`
- `normalize`
- `dot`
- `cross`

许多内置函数将标量或向量作为参数。对于向量参数，将按分量方式应用到函数，并返回计算后的结果向量。
Many of the built-in functions take either scalars or vectors as arguments. For vector arguments the function is applied component-wise and the resulting vector is returned.

abs

```
abs(x : Number) : Number
abs(x : vec2) : vec2
abs(x : vec3) : vec3
abs(x : vec4) : vec4
```

返回x的绝对值。
Returns the absolute value of x.

```
{
    "show" : "abs(${temperature}) > 20.0"
}
```

sqrt

```
sqrt(x : Number) : Number
sqrt(x : vec2) : vec2
sqrt(x : vec3) : vec3
sqrt(x : vec4) : vec4
```

当 `x >= 0`返回x的平方根。 当`x < 0`时返回 `NaN`。
Returns the square root of `x` when `x >= 0`. Returns `NaN` when `x < 0`.

```
{
    "color" : {
        "conditions" : [
            ["${temperature} >= 0.5", "color('#00FFFF')"],
            ["${temperature} >= 0.0", "color('#FF00FF')"]
        ]
    }
}
```

cos(余弦)

```
cos(angle : Number) : Number
cos(angle : vec2) : vec2
cos(angle : vec3) : vec3
cos(angle : vec4) : vec4
```

返回`angle`以弧度为单位的余弦值。
Returns the cosine of `angle` in radians.

```
{
    "show" : "cos(${Angle}) > 0.0"
}
```

sin（正弦）

```
sin(angle : Number) : Number
sin(angle : vec2) : vec2
sin(angle : vec3) : vec3
sin(angle : vec4) : vec4
```

返回`angle`以弧度为单位的正弦值。
Returns the sine of `angle` in radians.

```
{
    "show" : "sin(${Angle}) > 0.0"
}
```

tan（正切）

```
tan(angle : Number) : Number
tan(angle : vec2) : vec2
tan(angle : vec3) : vec3
tan(angle : vec4) : vec4
```

返回`angle`以弧度为单位的正切值。
Returns the tangent of `angle` in radians.

```
{
    "show" : "tan(${Angle}) > 0.0"
}
```

acos（反余弦）

```
acos(angle : Number) : Number
acos(angle : vec2) : vec2
acos(angle : vec3) : vec3
acos(angle : vec4) : vec4
```

返回`angle`以弧度为单位的反余弦值。
Returns the arccosine of `angle` in radians.

```
{
    "show" : "acos(${Angle}) > 0.0"
}
```

asin（反正弦）

```
asin(angle : Number) : Number
asin(angle : vec2) : vec2
asin(angle : vec3) : vec3
asin(angle : vec4) : vec4
```

返回`angle`以弧度为单位的反正弦值。
Returns the arcsine of `angle` in radians.

```
{
    "show" : "asin(${Angle}) > 0.0"
}
```

atan（反正切）

```
atan(angle : Number) : Number
atan(angle : vec2) : vec2
atan(angle : vec3) : vec3
atan(angle : vec4) : vec4
```

返回`angle`以弧度为单位的反正切值。
Returns the arctangent of `angle` in radians.

```
{
    "show" : "atan(${Angle}) > 0.0"
}
```

atan2

```
atan2(y : Number, x : Number) : Number
atan2(y : vec2, x : vec2) : vec2
atan2(y : vec3, x : vec3) : vec3
atan2(y : vec4, x : vec4) : vec4
```

返回`y`和`x`商的反正切。
Returns the arctangent of the quotient of `y` and `x`.

```
{
    "show" : "atan2(${GridY}, ${GridX}) > 0.0"
```

https://github.com/ElevenIjusee

```
}
```

radians（弧度）

```
radians(angle : Number) : Number
radians(angle : vec2) : vec2
radians(angle : vec3) : vec3
radians(angle : vec4) : vec4
```

将`angle`从角度转换为弧度。
Converts `angle` from degrees to radians.

```
{
    "show" : "radians(${Angle}) > 0.5"
}
```

degrees（角度）

```
degrees(angle : Number) : Number
degrees(angle : vec2) : vec2
degrees(angle : vec3) : vec3
degrees(angle : vec4) : vec4
```

将`angle`从弧度转为角度。
Converts `angle` from radians to degrees.

```
{
    "show" : "degrees(${Angle}) > 45.0"
}
```

sign（符号函数）

```
sign(x : Number) : Number
sign(x : vec2) : vec2
sign(x : vec3) : vec3
sign(x : vec4) : vec4
```

`x`为正时返回1.0，`x`为0时返回0.0,`x`为负返回-1.0。
Returns 1.0 when `x` is positive, 0.0 when `x` is zero, and -1.0 when `x` is negative.

```
{
    "show" : "sign(${Temperature}) * sign(${Velocity}) === 1.0"
}
```

floor（向下取整，向下舍入）

```
floor(x : Number) : Number
floor(x : vec2) : vec2
floor(x : vec3) : vec3
floor(x : vec4) : vec4
```

返回小于或等于最接近x的整数。
Returns the nearest integer less than or equal to x.

```
{
    "show" : "floor(${Position}) === 0"
}
```

ceil（向上取整）

```
ceil(x : Number) : Number
ceil(x : vec2) : vec2
ceil(x : vec3) : vec3
ceil(x : vec4) : vec4
```

返回大于或者等于x的最小整数。
Returns the nearest integer greater than or equal to x.

```
{
    "show" : "ceil(${Position}) === 1"
}
```

round（四舍五入取整）

```
round(x : Number) : Number
round(x : vec2) : vec2
round(x : vec3) : vec3
round(x : vec4) : vec4
```

返回最接近的整数x。分数为 0.5 的数字将按实现定义的方向舍入。
Returns the nearest integer to x. A number with a fraction of 0.5 will round in an implementation-defined direction.

```
{
    "show" : "round(${Position}) === 1"
}
```

exp

```
exp(x : Number) : Number
exp(x : vec2) : vec2
exp(x : vec3) : vec3
exp(x : vec4) : vec4
```

返回e的x次幂（次方），其中e是欧拉常数，约为`2.71828`。
Returns e to the power of x, where e is Euler's constant, approximately `2.71828`.

```
{
    "show" : "exp(${Density}) > 1.0"
}
```

log

```
log(x : Number) : Number
log(x : vec2) : vec2
log(x : vec3) : vec3
log(x : vec4) : vec4
```

返回x的自然对数（以e为底）。
Returns the natural logarithm (base e) of x.

```
{
    "show" : "log(${Density}) > 1.0"
}
```

exp2

```
exp2(x : Number) : Number
exp2(x : vec2) : vec2
```

```
exp2(x : vec3) : vec3
exp2(x : vec4) : vec4
```

返回2的x次方（次幂）。
Returns 2 to the power of x.

```
{
    "show" : "exp2(${Density}) > 1.0"
}
```

log2

```
log2(x : Number) : Number
log2(x : vec2) : vec2
log2(x : vec3) : vec3
log2(x : vec4) : vec4
```

返回已2为底的x的对数。
Returns the base 2 logarithm of x.

```
{
    "show" : "log2(${Density}) > 1.0"
}
```

fract（去小数）

```
fract(x : Number) : Number
fract(x : vec2) : vec2
fract(x : vec3) : vec3
fract(x : vec4) : vec4
```

返回x的小数部分。相当于`x - floor(x)`。
Returns the fractional part of x. Equivalent to `x - floor(x)`.

```
{
    "color" : "color() * fract(${Density})"
}
```

pow

```
pow(base : Number, exponent : Number) : Number
pow(base : vec2, exponent : vec2) : vec2
pow(base : vec3, exponent : vec3) : vec3
pow(base : vec4, exponent : vec4) : vec4
```

返回`base`的`exponent`次方（次幂）。
Returns `base` raised to the power of `exponent`.

```
{
    "show" : "pow(${Density}, ${Temperature}) > 1.0"
}
```

min

```
min(x : Number, y : Number) : Number
min(x : vec2, y : vec2) : vec2
min(x : vec3, y : vec3) : vec3
min(x : vec4, y : vec4) : vec4
```

```
min(x : Number, y : Number) : Number
min(x : vec2, y : Number) : vec2
min(x : vec3, y : Number) : vec3
min(x : vec4, y : Number) : vec4
```

返回`x`和`y`中的较小者。
Returns the smaller of `x` and `y`.

```
{
    "show" : "min(${Width}, ${Height}) > 10.0"
}
```

max

```
max(x : Number, y : Number) : Number
max(x : vec2, y : vec2) : vec2
max(x : vec3, y : vec3) : vec3
max(x : vec4, y : vec4) : vec4
max(x : Number, y : Number) : Number
max(x : vec2, y : Number) : vec2
max(x : vec3, y : Number) : vec3
max(x : vec4, y : Number) : vec4
```

返回 x和y中较大者。
Returns the larger of x and y.

```
{
    "show" : "max(${Width}, ${Height}) > 10.0"
}
```

clamp

```
clamp(x : Number,  min : Number, max : Number) : Number
clamp(x : vec2,  min : vec2, max : vec2) : vec2
clamp(x : vec3,  min : vec3, max : vec3) : vec3
clamp(x : vec4,  min : vec4, max : vec4) : vec4
```

```
clamp(x : Number,  min : Number, max : Number) : Number
clamp(x : vec2,  min : Number, max : Number) : vec2
clamp(x : vec3,  min : Number, max : Number) : vec3
clamp(x : vec4,  min : Number, max : Number) : vec4
```

将x的大小限制在min和max之间。
Constrains x to lie between min and max.

```
{
    "color" : "color() * clamp(${temperature}, 0.1, 0.2)"
}
```

mix

```
mix(x : Number,  y : Number, a : Number) : Number
mix(x : vec2,  y : vec2, a : vec2) : vec2
mix(x : vec3,  y : vec3, a : vec3) : vec3
mix(x : vec4,  y : vec4, a : vec4) : vec4
mix(x : Number,  y : Number, a : Number) : Number
mix(x : vec2,  y : vec2, a : Number) : vec2
mix(x : vec3,  y : vec3, a : Number) : vec3
mix(x : vec4,  y : vec4, a : Number) : vec4
```

计算' x '和' y '的线性内插值。
Computes the linear interpolation of `x` and `y`.

```
{
    "show" : "mix(20.0, ${Angle}, 0.5) > 25.0"
}
```

length

```
length(x : Number) : Number
length(x : vec2) : vec2
length(x : vec3) : vec3
length(x : vec4) : vec4
```

计算向量`x`的长度，即向量分量平方之和的平方根。
Computes the length of vector `x`, i.e., the square root of the sum of the squared components. If `x` is a number, `length` returns `x`.

```
{
    "show" : "length(${Dimensions}) > 10.0"
}
```

distance

```
distance(x : Number, y : Number) : Number
distance(x : vec2, y : vec2) : vec2
distance(x : vec3, y : vec3) : vec3
distance(x : vec4, y : vec4) : vec4
```

计算两点`x`和之间的距离`y`，即 `length(x - y)`。
Computes the distance between two points `x` and `y`, i.e., `length(x - y)`.

```
{
    "show" : "distance(${BottomRight}, ${UpperLeft}) > 50.0"
}
```

normalize

```
normalize(x : Number) : Number
normalize(x : vec2) : vec2
normalize(x : vec3) : vec3
normalize(x : vec4) : vec4
```

返回一个平行于`x`且长度为1.0的向量。当`x`是一个数字时，`normalize`返回1.0。
Returns a vector with length 1.0 that is parallel to `x`. When `x` is a number, `normalize` returns 1.0.

```
{
    "show" : "normalize(${RightVector}, ${UpVector}) > 0.5"
}
```

dot(点乘/点积)

```
dot(x : Number, y : Number) : Number
dot(x : vec2, y : vec2) : vec2
dot(x : vec3, y : vec3) : vec3
dot(x : vec4, y : vec4) : vec4
```

计算`x`和`y`的点乘。
Computes the dot product of `x` and `y`.

```
{
    "show" : "dot(${RightVector}, ${UpVector}) > 0.5"
}
```

cross（叉乘/叉积）

```
cross(x : vec3, y : vec3) : vec3
```

计算 `x` 和 `y` 的叉乘。这个函数只接受`vec3`参数。
Computes the cross product of `x` and `y`. This function only accepts `vec3` arguments.

```
{
    "color" : "vec4(cross(${RightVector}, ${UpVector}), 1.0)"
}
```

Point Cloud(点云)

一个点云Point Cloud是一个由许多点组成的集合，可以像其他feature一样样式化。除了计算一个点的color和show属性，点云还可以计算pointSize，或者每个点的像素大小。默认pointSize是1.0。
A Point Cloud is a collection of points that may be styled like other features. In addition to evaluating a point's color and show properties, a Point Cloud style may evaluate pointSize, or the size of each point in pixels. The default pointSize is 1.0.

```
{
    "color" : "color('red')",
    "pointSize" : "${Temperature} * 0.5"
}
```

实际应用中，可能会将pointSize的值限制到系统支持的点大小范围。例如，当渲染POINTS时，WebGL渲染器将查询ALIASED_POINT_SIZE_RANGE以获得系统限制。确保pointSize的值为1.0必须得到支持。
Implementations may clamp the evaluated pointSize to the system's supported point size range. For example, WebGL renderers may query ALIASED_POINT_SIZE_RANGE to get the system limits when rendering with POINTS. A pointSize of 1.0 must be supported.
点云的样式还可以从Feature Table中引用包括位置，颜色和法线的语义，以允许更灵活的对源数据进行样式设置。
Point Cloud styles may also reference semantics from the Feature Table including position, color, and normal to allow for more flexible styling of the source data.

- ${POSITION}是一个vec3格式储存的xyz 笛卡尔点坐标，这个坐标是在RTC_CENTER和tile的平移变换（transform）之前的位置。当位置被量化后，${POSITION}指的是应用QUANTIZED_VOLUME_SCALE之后，但应用QUANTIZED_VOLUME_OFFSET之前的位置。
- ${POSITION} is a vec3 storing the xyz Cartesian coordinates of the point before the RTC_CENTER and tile transform are applied. When the positions are quantized, ${POSITION} refers to the position after the QUANTIZED_VOLUME_SCALE is applied, but before QUANTIZED_VOLUME_OFFSET is applied.
- ${POSITION_ABSOLUTE}是一个vec3格式储存的xyz 笛卡尔点坐标。这个坐标应用在RTC_CENTER和tile平移变换之后。当位置被量化后，${POSITION_ABSOLUTE}指的是应用QUANTIZED_VOLUME_SCALE, QUANTIZED_VOLUME_OFFSET和tile平移变换后的位置。
- ${POSITION_ABSOLUTE} is a vec3 storing the xyz Cartesian coordinates of the point after the RTC_CENTER and tile transform are applied. When the positions are quantized, ${POSITION_ABSOLUTE} refers to the position after the QUANTIZED_VOLUME_SCALE, QUANTIZED_VOLUME_OFFSET, and tile transform are applied.
- ${COLOR}的计算结果为存储点rgba颜色的Color。当Feature Table的颜色格式是RGB或者 RGB565时，${COLOR}.alpha的值为1.0。如果没有定义颜色，${COLOR} 将计算为专用的默认颜色。
- ${COLOR} evaluates to a Color storing the rgba color of the point. When the Feature Table's color semantic is RGB or RGB565, ${COLOR}.alpha is 1.0. If no color semantic is defined, ${COLOR} evaluates to the application-specific default color.
- ${NORMAL}是以vec3格式存储的笛卡尔坐标法线，这个法线应用在tile平移变换之前。当法线是八进制编码时，${NORMAL}指的是已解码的法线。如果Feature Table中没有定义法线的语义，${NORMAL}的计算结果为undefined。

- `${NORMAL}` is a `vec3` storing the normal, in Cartesian coordinates, of the point before the tile transform is applied. When normals are oct-encoded, `${NORMAL}` refers to the decoded normal. If no normal semantic is defined in the Feature Table, `${NORMAL}` evaluates to `undefined`.

For example:

```
{
    "color" : "${COLOR} * color('red')'",
    "show" : "${POSITION}.x > 0.5",
    "pointSize" : "${NORMAL}.x > 0 ? 2 : 1"
}
```

> **实际应用的注意事项:** 点云样式引擎通常使用shader（GLSL）实现，但是一些feature的表达式语句是不可能在GLSL中实现的。这些features包括： **Implementation Note:** Point cloud styling engines may often use a shader (GLSL) implementation, however some features of the expression language are not possible in pure a GLSL implementation. Some of these features include:
>
> - 结果为`isNaN` 和`isFinite` （GLSL 2.0+ 分别支持`isnan`和`isinf`这些功能）；
> - Evaluation of `isNaN` and `isFinite` (GLSL 2.0+ supports `isnan` and `isinf` for these functions respectively)
> - 类型为`null` 和`undefined`
> - The types `null` and `undefined`
> - 字符串，包括访问对象属性(`color()['r']`)和批处理表的值。
> - Strings, including accessing object properties (`color()['r']`) and batch table values
> - 正则表达式
> - Regular expressions
> - 长度不为 2、3 或 4 的数组
> - Arrays of lengths other than 2, 3, or 4
> - 不同类型比较（例如`1.0 === false`)
> - Mismatched type comparisons (e.g. `1.0 === false`)
> - 数组索引越界
> - Array index out of bounds

File extension and MIME type（文件扩展名和MIME类型）

Tileset样式文件使用 `.json`作为扩展名，`application/json`作为mime类型.
Tileset styles use the `.json` extension and the `application/json` mime type.

Property reference（属性参考）

- `style`
  - `boolean expression`（布尔表达式）
  - `color expression`（颜色表达式）
  - `conditions`
    - `condition`
  - `expression`（表达式）
  - `meta`（元）

  - `number expression`(数字表达式)
- `Point Cloud Style`（点云样式）

---

style

A 3D Tiles style.

Properties

	Type	Description	Required
defines	`object`	`表达式（expression）`映射到以变量名称为key的字符串字典对象，其可以在整个style中引用。如果一个表达式引用了一个已定义的变量，它会被相应表达式的计算结果替换。	No
show	`boolean`, `string`, `object`	一个用于确定feature是否应显示的`布尔表达式（boolean expression）` 或 `conditions` 属性。	No, default: `true`
color	`string`, `object`	`颜色表达式（color expression）`或者`conditions`属性。用来决定是什么新颜色与feature固有的颜色进行混合	No, default: `color('#FFFFFF')`
meta	`object`	一个`元（meta）`对象。决定了该feature那些非视觉属性的值	No

不允许附加属性。
Additional properties are not allowed.

style.defines

`表达式（expression）`映射到变量名称键的字符串字典对象，可以在整个样式中引用。如果一个表达式引用了一个定义的变量，它会被相应表达式的计算结果替换。
A dictionary object of `expression` strings mapped to a variable name key that may be referenced throughout the style. If an expression references a defined variable, it is replaced with the evaluated result of the corresponding expression.

- **Type**: `object`
- **Required**: No
- **Type of each property**: `string`

style.show

一个用于确定feature是否应显示的`布尔表达式（boolean expression）` 或 `conditions` 属性。
`boolean expression` or `conditions` property which determines if a feature should be shown.

- **Type**: `boolean`, `string`, `object`
- **Required**: No, default: `true`

style.color

颜色表达式（color expression）或者conditions属性。用来决定是什么新颜色与feature固有的颜色进行混合
A color expression or conditions property which determines the color blended with the feature's intrinsic color.

- **Type**: string, object
- **Required**: No, default: color('#FFFFFF')

style.meta

一个元（meta）对象。决定了该feature那些非视觉属性的值。
A meta object which determines the values of non-visual properties of the feature.

- **Type**: object
- **Required**: No
- **Type of each property**: string

---

boolean expression（布尔表达式）

具有 3D Tiles 样式表达式的布尔值或字符串，其计算结果为布尔值。请参阅表达式Expressions
A boolean or string with a 3D Tiles style expression that evaluates to a boolean. See Expressions.

- **JSON schema**: style.booleanExpression.schema.json

---

color expression（颜色表达式）

最终结果为Color的3D Tiles样式表达式，参见表达式Expressions。
3D Tiles style expression that evaluates to a Color. See Expressions.

- **JSON schema**: style.colorExpression.schema.json

---

conditions（条件）

一系列按顺序计算的条件，就像一系列 if...else 语句。
A series of conditions evaluated in order, like a series of if...else statements that result in an expression being evaluated.

Properties

	Type	Description	Required
conditions	array []	按照顺序计算的一系列布尔条件。对于第一个计算结果为true的值，它的值（即result，也是一个表达式）将被计算并返回。结果表达式必须都是相同的类型。如果没有条件为真，则结果为undefined。当条件为undefined，null或空对象时，结果也是undefined。	No

不允许附加属性。
Additional properties are not allowed.

conditions.conditions

按照顺序计算的一系列布尔条件。对于第一个计算结果为true的值，它的值（即`result`，也是一个表达式）将被计算并返回。结果表达式必须都是相同的类型。如果没有条件为真，则结果为`undefined`。当条件为`undefined`，`null`或空对象时，结果也是`undefined`。
A series of boolean conditions evaluated in order. For the first one that evaluates to true, its value, the 'result' (which is also an expression), is evaluated and returned. Result expressions must all be the same type. If no condition evaluates to true, the result is `undefined`. When conditions is `undefined`, `null`, or an empty object, the result is `undefined`.

- **Type**: `array []`
- **Required**: No

---

condition

将条件为真的结果进行计算。包含两个表达式的数组。如果第一个表达式被求值并且结果为`true`，那么第二个表达式被求值并作为条件的结果返回。
An `expression` evaluated as the result of a condition being true. An array of two expressions. If the first expression is evaluated and the result is `true`, then the second expression is evaluated and returned as the result of the condition.

- **JSON schema**: `style.conditions.condition.schema.json`

---

expression

一个有效的3D Tile样式表达式。参阅 Expressions。
A valid 3D Tiles style expression. See Expressions.

- **JSON schema**: `style.expression.schema.json`

---

meta（元）

一系列属性名称和表达式`expression`来计算该属性的值。
A series of property names and the `expression` to evaluate for the value of that property.

允许附加属性。
Additional properties are allowed.

- **每个属性的类型**: `expression`
- **Type of each property**: `expression`

---

number expression（数字表达式）

计算结果为数字的 3D Tiles 样式表达式。参见Expressions.
3D Tiles style expression that evaluates to a number. See Expressions.

- **JSON schema**: `style.numberExpression.schema.json`

---

https://github.com/ElevenIjusee

Point Cloud Style（点云样式）

附加属性为点云的3D Tiles样式表达式。
A 3D Tiles style with additional properties for Point Clouds.

Properties

	Type	Description	Required
defines	`object`	表达式（`expression`）映射到以变量名称为key的字符串字典对象，其可以在整个style中引用。如果一个表达式引用了一个已定义的变量，它会被相应表达式的计算结果替换。	No
show	`boolean`, `string`, `object`	一个用于确定feature是否应显示的布尔表达式（`boolean expression`） 或 `conditions` 属性。	No, default: `true`
color	`string`, `object`	颜色表达式（`color expression`）或者`conditions`属性。用来决定是什么新颜色与feature固有的颜色进行混合	No, default: `color('#FFFFFF')`
meta	`object`	一个元（`meta`）对象。决定了该feature那些非视觉属性的值	No
pointSize	`number`, `string`, `object`	一个数字表达式`number expression` 或者 `conditions`属性，它决定了点的大小（以像素为单位）。	No, default: `1`

不允许附加属性。
Additional properties are not allowed.

PointCloudStyle.defines

(#reference-expression)映射到以变量名称为key的字符串字典对象，其可以在整个style中引用。如果一个表达式引用了一个已定义的变量，它会被相应表达式的计算结果替换。
A dictionary object of `expression` strings mapped to a variable name key that may be referenced throughout the style. If an expression references a defined variable, it is replaced with the evaluated result of the corresponding expression.

- **Type**: `object`
- **Required**: No
- **Type of each property**: `string`

PointCloudStyle.show

一个用于确定feature是否应显示的布尔表达式（`boolean expression`） 或 `conditions` 属性。
A `boolean expression` or `conditions` property which determines if a feature should be shown.

- **Type**: `boolean`, `string`, `object`
- **Required**: No, default: `true`

PointCloudStyle.color

`颜色表达式（color expression）`或者`conditions`属性。用来决定是什么新颜色与feature固有的颜色进行混合。
A `color expression` or `conditions` property which determines the color blended with the feature's intrinsic color.

- **Type**: `string`, `object`
- **Required**: No, default: `color('#FFFFFF')`

PointCloudStyle.meta

一个`元（meta）`对象。决定了该feature那些非视觉属性的值。
A `meta` object which determines the values of non-visual properties of the feature.

- **Type**: `object`
- **Required**: No
- **Type of each property**: `string`

PointCloudStyle.pointSize

一个`数字表达式number expression`或者 `conditions`属性，它决定了点的大小（以像素为单位）。
A `number expression` or `conditions` property which determines the size of the points in pixels.

- **Type**: `number`, `string`, `object`
- **Required**: No, default: `1`