**Vega小结**

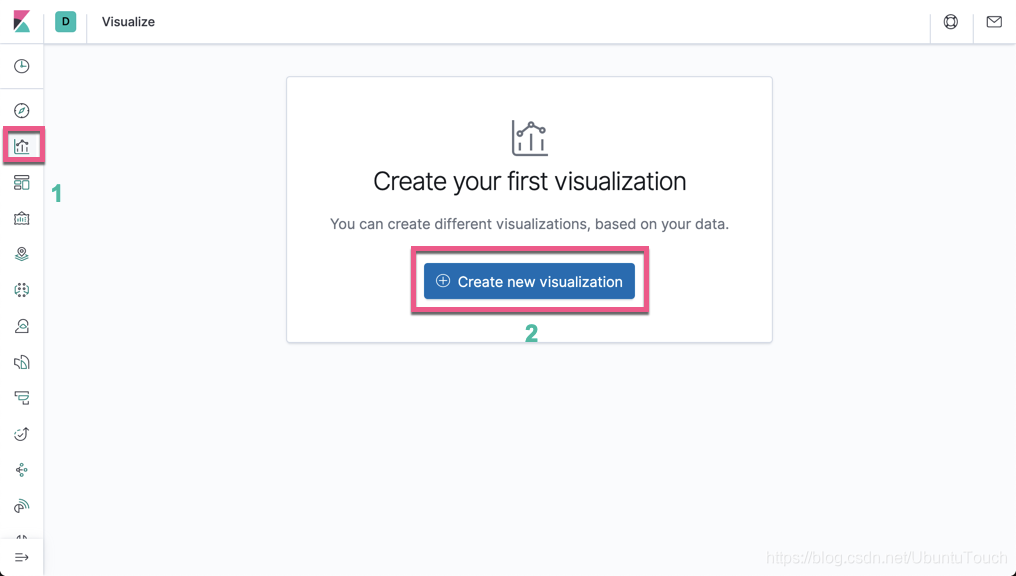
**一 概念和创建方法**

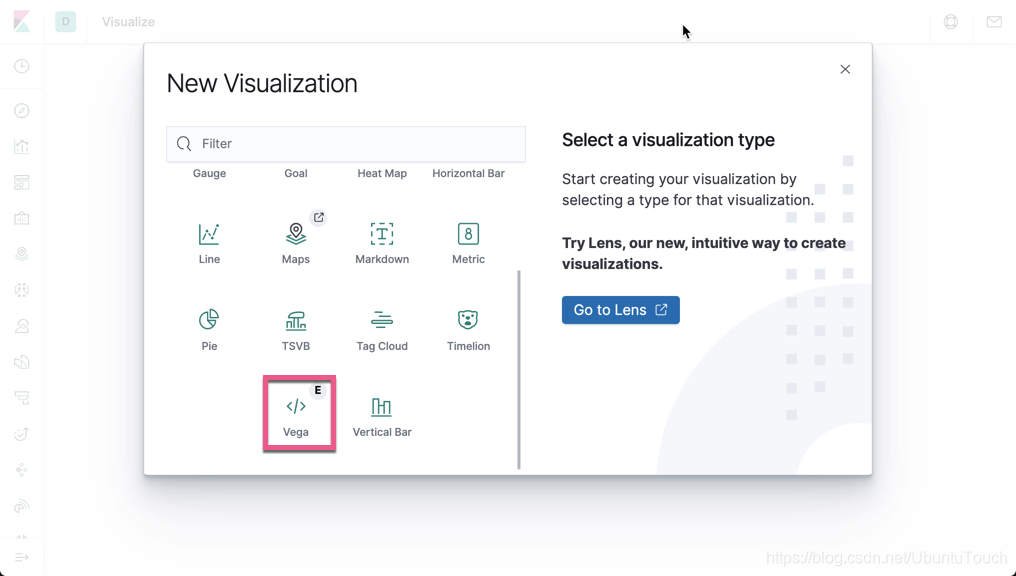
Vega是Elastic search中kibana模块中一种可定制化的绘图工具。

Vega 声明式语法是一种可视化数据的强大方法。可视化内容以 JSON 描述，并使用 HTML5 Canvas 或 SVG 生成交互式视图。 它是Kibana 6.2中的一项新功能，我们可以使用 Elasticsearch 数据构建丰富的Vega 和 Vega-Lite 可视化

Vega规范是描述交互式可视化设计的JSON对象。规范包括基本属性(比如视图的宽度和高度)和定义的数据可视化,尺度数据值映射到视觉值,轴和传说,想象这些尺度,地图绘制地图,预测图形标志如矩形、线条,视觉上代表数据和符号,以及处理用户输入和相应修改可视化的信号。

Vega一般是在Elasticsearch的web页面点击左边的窗口中的“visualize”，然后再点击“create visualization”创建视图，再点击选择“customize visualization”即可。





**二 基本写法**

1. **基础知识**

Vega规范用JavaScript对象表示法(JSON)定义了交互式可视化。

下面是一个Vega规范的基本大纲。完整的规范包括数据、比例、轴、标记等属性的适当子集的定义。

{

**"$schema"**: "https://vega.github.io/schema/vega/v5.json",

**"description"**: "A specification outline example.",

**"width"**: 500,

**"height"**: 200,

**"padding"**: 5,

**"autosize"**: "pad",

**"signals"**: [],

**"data"**: [],

**"scales"**: [],

**"projections"**: [],

**"axes"**: [],

**"legends"**: [],

**"marks"**: []}

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| $schema | [URL](https://vega.github.io/vega/docs/types/" \l "URL) | Vega模式的URL。 |
| description | [String](https://vega.github.io/vega/docs/types/" \l "String) | 可视化的文本描述。在≥5.10版本中，描述决定了Vega视图的容器元素的aria-label属性。 |
| background | [Color](https://vega.github.io/vega/docs/types/" \l "Color) | [Signal](https://vega.github.io/vega/docs/types/" \l "Signal) | 整个视图的背景色(默认为透明)。如果signal-value≥5.10，则使用所提供的表达式作为底层背景信号定义的更新属性。 |
| width | [Number](https://vega.github.io/vega/docs/types/" \l "Number) | [Signal](https://vega.github.io/vega/docs/types/" \l "Signal) | 数据矩形的像素宽度。如果signal-value≥5.10，则使用所提供的表达式作为底层宽度信号定义的更新属性。 |
| height | [Number](https://vega.github.io/vega/docs/types/" \l "Number) | [Signal](https://vega.github.io/vega/docs/types/" \l "Signal) | 数据矩形的像素高度。如果signal-value≥5.10，则使用所提供的表达式作为底层高度信号定义的更新属性。 |
| padding | [Number](https://vega.github.io/vega/docs/types/" \l "Number) | [Object](https://vega.github.io/vega/docs/types/" \l "Object) | [Signal](https://vega.github.io/vega/docs/types/" \l "Signal) | 在可视化周围添加像素填充。如果是一个数字，则指定所有边的填充。如果是一个对象，值的格式应该是{"left": 5， "top": 5， "right": 5， "bottom": 5}。填充在自动调整布局完成后应用。如果signal-value≥5.10，所提供的表达式将用作底层填充信号定义的update属性，并应计算为填充对象或数字。 |
| autosize | [String](https://vega.github.io/vega/docs/types/" \l "String) | [Autosize](https://vega.github.io/vega/docs/specification/" \l "autosize) | [Signal](https://vega.github.io/vega/docs/types/" \l "Signal) | 设置应该如何确定可视化大小。如果是字符串，则应该是pad(默认值)、fit、fit-x、fit-y或none。对象值还可以指定内容大小调整和自动大小调整的参数。更多信息请参见下面的自动大小部分。如果signal-value≥5.10，所提供的表达式将用作底层autosize信号定义的update属性，并应计算为一个完整的autosize对象。 |
| config | [Config](https://vega.github.io/vega/docs/config) | 为标记、轴和图例使用默认值的配置设置。 |
| signals | [Signal](https://vega.github.io/vega/docs/signals)[ ] | 信号是将可视化参数化的动态变量。 |
| data | [Data](https://vega.github.io/vega/docs/data)[ ] | 数据集定义和转换定义要加载的数据以及如何处理数据。 |
| scales | [Scale](https://vega.github.io/vega/docs/scales)[ ] | 比例尺将数据值(数字、日期、类别等)映射到可视值(像素、颜色、大小)。 |
| projections | [Projection](https://vega.github.io/vega/docs/projections)[ ] | 地图投影地图(经度，纬度)对投影(x, y)坐标。 |
| axes | [Axis](https://vega.github.io/vega/docs/axes)[ ] | 坐标轴使空间尺度映射可视化。 |
| legends | [Legend](https://vega.github.io/vega/docs/legends)[ ] | 图例可视化可视值(如颜色、形状和大小)的比例映射。 |
| title | [Title](https://vega.github.io/vega/docs/title) | 标题文字描述可视化。 |
| marks | [Mark](https://vega.github.io/vega/docs/marks)[ ] | 图形标记可视化地使用几何原语(如矩形、线和绘图符号)编码数据。 |
| encode | [Encode](https://vega.github.io/vega/docs/marks/" \l "encode) | 用于表示图表数据矩形的顶级组标记的可视属性的编码指令。例如，这可以用来设置绘图区域的背景填充颜色，而不是整个视图。 |
| usermeta | [Object](https://vega.github.io/vega/docs/types/" \l "Object) | 将被Vega解析器忽略的可选元数据。 |

**Autosize**

可以通过各种方式调整(和调整)Vega视图的大小。如果是一个对象，该值的格式应该是{"type": "pad"， "resize": true}，其中type是自动大小字符串之一，resize是布尔值，指示每次更新是否需要重新计算自动大小布局。

| **Name** | **Type** | **Description** |
| --- | --- | --- |
| type | [String](https://vega.github.io/vega/docs/types/" \l "String) | **必需的。大小调整格式类型。“pad”(默认)之一,“适合”,“fit-x”、“fit-y”或“none”。** |
| resize | [Boolean](https://vega.github.io/vega/docs/types/" \l "Boolean) | 一个布尔标志，指示在每次视图更新时是否应该重新计算自动大小布局。默认值(false)导致布局在初始化时执行一次，并响应高度和/或宽度信号的变化(请参阅这里了解更多大小调整逻辑)。否则，布局将保持稳定。要从外部强制调整大小，请使用“视图”。调整API方法。 |
| contains | [String](https://vega.github.io/vega/docs/types/" \l "String) | 确定应该如何执行大小计算，内容(默认)或填充。默认设置(内容)将宽度和高度设置解释为数据矩形(绘图)尺寸，然后向其添加填充。相反，填充设置包括视图大小计算中的填充，因此宽度和高度设置指示视图的总预期大小。 |

**各个模块的概念和使用参数（常用\*/不常用）**

**Config**

配置对象定义默认可视值来设置可视化主题。

Vega解析器接受一个JSON配置文件，该文件为各种可视编码选择定义默认设置。不同的配置文件可以用于“主题”图表的自定义外观和感觉。

具体的config使用方法参见：

<https://vega.github.io/vega/docs/config/>

**Data\***

Vega使用的基本数据模型是表格数据，类似于电子表格或数据库表。假设单个数据集包含一个记录集合(或“行”)，其中可能包含任意数量的命名数据属性(字段或“列”)。记录是使用标准JavaScript对象建模的。

如果输入数据只是一个基本值数组，那么Vega将每个值映射到新对象的数据属性。例如[5,3,8,1]被加载为:

[ {**"data"**: 5}, {**"data"**: 3}, {**"data"**: 8}, {**"data"**: 1} ]

在摄取数据时，Vega还为每个数据对象分配一个唯一的id属性，通过自定义Symbol进行访问。因此，id属性不能通过字符串键访问，也不能被枚举，尽管您可以在JavaScript控制台中检查数据对象时观察id值。

数据集可以通过定义内联数据或提供一个URL来直接指定。或者，当图表实例化或发布流更新时，可以使用View API在运行时动态绑定数据。从URL加载数据将取决于您的运行时环境的策略(例如，跨源请求规则)。

**数据的格式**

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| name | [String](https://vega.github.io/vega/docs/types/" \l "String) | **必需的。数据集的唯一名称。** |
| format | [Format](https://vega.github.io/vega/docs/data/" \l "format) | 指定解析数据文件或值的格式的对象。 |
| source | [String](https://vega.github.io/vega/docs/types/" \l "String) | [String](https://vega.github.io/vega/docs/types/" \l "String)[ ] | 用作此数据集源的一个或多个数据集的名称。source属性与转换管道结合使用非常有用，可以派生新数据。如果为字符串值，则指示源数据集的名称。如果是数组值，则指定应该合并(合并)在一起的数据源名称的集合。 |
| url | [String](https://vega.github.io/vega/docs/types/" \l "String) | 用于加载数据集的URL。使用format属性确保正确解析加载的数据。如果没有指定format属性，则假定数据是面向行的JSON格式。 |
| values | [Any](https://vega.github.io/vega/docs/types/" \l "Any) | 完整的数据集，内联包含。values属性允许在规范本身中直接包含数据。虽然最常见的是对象数组，但也可能使用其他数据类型(如CSV字符串)，这取决于格式设置。 |
| async | [Boolean](https://vega.github.io/vega/docs/types/" \l "Boolean) | ≥5.9一个布尔标志(默认为false)，指示动态数据加载或重新格式化是否应该异步进行。如果为true，则数据流评估将完成，数据加载将在后台进行，数据流加载完成后将重新评估。如果为false，则数据流的计算将阻塞，直到加载完成，然后在相同的计算周期内继续。使用async可以允许同时加载多个动态数据集，同时仍然支持交互性。然而，使用async可能会导致在计算其余数据流时数据集仍然为空，从而可能影响下游的计算。 |
| on | [Trigger](https://vega.github.io/vega/docs/triggers)[ ] | 一个数组，用于插入、删除和切换数据值，或在满足触发条件时清除数据。更多信息请参见触发器参考。 |
| transform | [Transform](https://vega.github.io/vega/docs/transforms)[ ] | 要对输入数据执行的转换数组。然后，转换管道的输出将成为该数据集的值。有关更多信息，请参阅转换参考。 |

**Transforms**

转换处理数据流以过滤数据、计算新的字段或派生新的数据流。转换通常在数据定义的转换数组中指定。此外，不筛选或生成新数据对象的转换可以在标记定义的转换数组中使用，以指定编码后转换。

|  |
| --- |
| **Basic Transforms** |
| **aggregate—对数据流进行分组和汇总。**  bin -离散数值到统一的箱子。  collect——收集流中的所有数据对象并对其进行排序。  countpattern—计算文本字符串中模式的频率。  cross-执行数据流本身的交叉乘积。  density-从概率分布中生成值。  dotbin -执行密度装箱的点状图建设。≥5.7  extent—计算数据流上的最小值和最大值。  **filter—使用谓词表达式过滤数据流。**  flatten—将数组类型字段映射为数据对象，每个数组项一个。≥3.1  fold-将选定的数据字段折叠成键和值属性。  **formula—使用公式表达式扩展带有派生字段的数据对象。**  identifier—为数据对象分配唯一的键值。  kde-估计平滑密度的数值值。≥5.4  impute—执行丢失值的输入。  joinaggregate—使用计算的聚合值扩展数据对象。  loess-拟合一个平滑的趋势线使用局部回归。≥5.4  lookup—通过在另一个流上查找键值来扩展数据对象。  pivot-枢轴独特的值到新的聚合字段。≥3.2  project—生成带有选定字段集的派生数据对象。  quantile——计算输入数据流上的采样分位数值。≥5.7  regression-拟合回归模型平滑和预测值。≥5.4  sample -在流中随机抽样数据对象。  sequence—生成一个包含数值序列的新流。  **timeunit—将日期时间值离散化到时间单位容器中。≥5.8**  window -计算超排序的组，包括排名和运行总数。 |

结合项目的使用情况，我们重点整理了aggregate/filter/formula/timeunit四个基本转换知识点。

|  |
| --- |
| **Aggregate Transform** |
| | **Property** | **Type** | **Description** | | --- | --- | --- | | groupby | [Field](https://vega.github.io/vega/docs/types/" \l "Field)[ ] | 要分组的数据字段。如果未指定，将使用包含所有数据对象的单个组。 | | fields | [Field](https://vega.github.io/vega/docs/types/" \l "Field)[ ] | 要计算聚合函数的数据字段。这个数组应该与操作对齐，并作为数组。如果没有指定字段和操作，默认情况下将使用计数聚合。 | | ops | [String](https://vega.github.io/vega/docs/types/" \l "String)[ ] | 应用于字段的聚合操作，如sum、average或count。如果没有指定操作，默认情况下将使用计数聚合。有关更多信息，请参阅聚合操作参考。 | | as | [String](https://vega.github.io/vega/docs/types/" \l "String)[ ] | 字段中每个聚合字段要使用的输出字段名。如果没有指定，名称将根据操作和字段名自动生成(例如，sum\_field, average\_field)。 | | cross | [Boolean](https://vega.github.io/vega/docs/types/" \l "Boolean) | 指示是否应该在聚合输出中包含所有groupby值的完整交叉积(默认为false)。如果设置为true，将考虑groupby字段值的所有可能组合，并为没有出现在数据本身中的组合生成并返回零计数组。跨积输出的行为就好像drop参数为false。在流式更新的情况下，如果观察到新的groupby字段值，输出组的数量将增加;所有先前的组将被保留。这个参数对于生成包括所有可能的数据分区的组的facet很有用。 | | drop | [Boolean](https://vega.github.io/vega/docs/types/" \l "Boolean) | 指示是否应该删除空(零计数)组(默认为true)。当数据流更新时(例如，响应交互式过滤)，聚合组可能变为空。默认情况下，将从输出中删除该组。然而，在某些情况下(如直方图)，可能希望保留空组。 | | key | [Field](https://vega.github.io/vega/docs/types/" \l "Field) | 一个可选的键字段，用于优化分组键计算。如果指定，则每个聚合单元的唯一键将不会从groupby字段本身生成，而是只使用这个单一键字段。在有多个分组字段的情况下，使用键有助于加快处理，但单个字段就足以区分每个聚合单元。例如，对于直方图来说，只在bin0属性上键入键会更快，当bin1属性(也作为一个groupby字段包含)包含与分组相关的冗余信息时，这是安全的。应该谨慎使用该参数，并且只有在确定关键字段唯一地区分所有分组字段值的组合时才使用该参数。 | |
| **Aggregate Transform** |
| | **Property** | **Type** | **Description** | | --- | --- | --- | | expr | [Expr](https://vega.github.io/vega/docs/types/" \l "Expr) | 必需的。用于过滤数据的谓词表达式。如果表达式的计算结果为false，则该数据对象将被筛选。 | |
| **Formula Transform** |
| | **Property** | **Type** | **Description** | | --- | --- | --- | | expr | [Expr](https://vega.github.io/vega/docs/types/" \l "Expr) | 必需的。计算导出值的公式表达式。 | | as | [String](https://vega.github.io/vega/docs/types/" \l "String) | 必需的。要在其中写入公式值的输出字段。 | | initonly | [Boolean](https://vega.github.io/vega/docs/types/" \l "Boolean) | 如果为真，则仅在第一次观察数据对象时计算公式。如果数据对象被修改，公式值将不会自动更新。默认为false。 | |
| **Timeunit Transform** |
| | **Property** | **Type** | **Description** | | --- | --- | --- | | field | [Field](https://vega.github.io/vega/docs/types/" \l "Field) | 必需的。日期-时间值的数据字段。 | | units | [String](https://vega.github.io/vega/docs/types/" \l "String)[ ] | 定义日期时间值应如何装箱的时间单位说明符数组。更多信息请参见时间单位参考。如果未指定，将根据值范围推断单位。 | | step | [Number](https://vega.github.io/vega/docs/types/" \l "Number) | 按单位提供的最小时间单位计算的各个垃圾箱之间的步长(默认为1)。如果单位未指定，此参数将被忽略。 | | timezone | [String](https://vega.github.io/vega/docs/types/" \l "String) | 要使用的时区，本地时区为“local”(默认值)之一，协调世界时为“utc”。 | | interval | [String](https://vega.github.io/vega/docs/types/" \l "String) | 一个布尔标志(默认为true)，指示是否同时输出开始和结束单元值。如果为false，则只将起始时间单位值写入输出。 | | extent | [Date](https://vega.github.io/vega/docs/types/" \l "Date)[ ] | 一个包含最小值和最大值的双元素数组，用于推断时间单元。如果未指定，将使用观测到的字段的最小值和最大值。该参数仅在未指定units参数时有效。 | | maxbins | [Number](https://vega.github.io/vega/docs/types/" \l "Number) | 为推断时间单元创建的最大箱数(默认为40)。当域被切成“很好”的四舍五入值时，通常会有更少的垃圾箱。该参数仅在未指定units参数时有效。 | | signal | [String](https://vega.github.io/vega/docs/types/" \l "String) | 如果定义了，则将计算时间单位规范绑定到具有给定名称的信号。被绑定的对象有单位(最小的时间单位)，单位(所有时间单位的数组)，步进，开始和停止属性。 | | as | [String](https://vega.github.io/vega/docs/types/" \l "String)[ ] | 用于写入起始时间和结束时间单位间隔值的输出字段。默认是["unit0"， "unit1"]。 | |

时间单元转换接受以下预定义的时间单元集合。单个单元值是以下字符串之一:

* "year" - 公历年。
* "quarter" - 间隔三个月，从1月、4月、7月和10月中的一个开始。
* "month" - 日历月(一月，二月等)。
* "date" - 日历上的某一天(1月1日，1月2日等)。
* "week" - Sunday-based周。一年的第一个星期天之前的日子被认为是在第0周，一年的第一个星期天是第1周的开始，第二个星期天是第2周，等等。
* "day" - 星期几(星期日、星期一等)。
* "dayofyear" - 一年中的一天(1,2，…，365，等等)。
* "hours" - 一天的时间(中午12点、凌晨1点等)。
* "minutes" - 一小时的分钟(12:00、12:01等)。
* "seconds" - 一分钟中的秒(12:00:00,12:00:01，等等)。
* "milliseconds" - 毫秒。

单元转换参数接受一个数组来表示所需的时间间隔。例如，["year"， "month"， "date"]表示对年、月和日期(但对小时、分钟或秒)时间敏感。说明符["month"， "date"]对月和日期敏感，但对年不敏感，这对于分拣时间值以只查看季节模式很有用。

**Triggers**

触发器允许在满足特定条件时动态更新数据集或标记项。当一个触发器表达式(通常引用一个或多个信号)求值为真值时，将应用一个或多个数据更新(插入、删除、切换和/或修改)。

触发器既可以用于更新数据集中的数据对象，也可以用于更新标记定义中标记项的属性(仅修改)。请注意，派生数据集不支持触发器;在派生数据集上定义的任何触发器都不起作用。

<https://vega.github.io/vega/docs/triggers/>

**Projection**

地图投影地图(经度，纬度)对投影(x, y)坐标。Vega使用投影来布局两个地理点(例如地图上的位置)，其中经度和纬度坐标是输入数据的一部分，以及使用GeoJSON格式表示的地理区域(例如国家和州)。

具体讲解如下链接所示：

<https://vega.github.io/vega/docs/projections/>

**Scales**

比例尺将数据值(数字、日期、类别等)映射到可视值(像素、颜色、大小)。尺度是数据可视化的一个基本构建块，因为它们决定了可视化编码的性质。Vega包括一系列连续和离散输入数据的尺度，并支持位置、形状、大小和颜色编码的映射。为了可视化比例，vega-lite规范可能包括轴或图例。

具体讲解如下链接所示：

<https://vega.github.io/vega/docs/scales/>

**Color schemes**

颜色方案为离散和连续颜色编码提供了一组命名的调色板。Vega提供了一系列直观的配色方案，其中许多最初是由Cynthia Brewer和ColorBrewer项目，或由Tableau Software的Maureen Stone创建的。要查看和设置默认颜色模式，请参阅Config文档。

具体讲解如下链接所示：

<https://vega.github.io/vega/docs/schemes/>

**Axes**

轴使用刻度、网格线和标签可视化空间尺度映射。Vage-lite目前支持笛卡儿坐标(直角)轴。与比例类似，轴既可以在规范的顶层定义，也可以作为组标记的一部分定义。

离散配色方案可以直接与具有离散(或离散)域的尺度一起使用，例如序数尺度、量化尺度和分位数尺度。连续配色方案可以直接与连续尺度(如线性、对数和根号尺度)一起使用，而且——通过指定方案计数属性——还可以用于生成离散配色方案。

具体讲解如下链接所示：

<https://vega.github.io/vega/docs/axes/>

**Legends**

图例可视化可视值(如颜色、形状和大小)的比例映射。与比例和轴类似，图例既可以在顶层可视化中定义，也可以在组标记的范围内定义。

用于指定图例的属性。图例接受一个或多个比例作为参数。必须指定至少一个大小、形状、填充、描边、stroke dash或不透明度属性。如果提供了多个尺度，它们必须共享相同的输入域。否则，图例的行为是未定义的。

具体讲解如下链接所示：

<https://vega.github.io/vega/docs/legends/>

**Title**

title指令将描述性标题添加到图表中。与刻度、轴和图例类似，标题可以在规范的顶层定义，也可以作为组标记的一部分定义。

具体讲解如下链接所示：

<https://vega.github.io/vega/docs/title/>

**Marks**

图形标记可视化地使用几何原语(如矩形、线和绘图符号)编码数据。标记是可视化的基本可视化构建块，提供基本形状，其属性可以根据后备数据设置。标记属性定义可以是简单的常量或数据字段，也可以使用尺度将数据值映射到可视值。

支持的标记类型有:

arc——圆弧，包括馅饼和甜甜圈切片。

area -水平或垂直排列的填充区域。

image—图像，包括图标或照片。

group -用于其他标记的容器，用于子图。

line-有笔画的线条，常用于显示随时间的变化。

path—使用SVG路径语法定义的任意路径或多边形。

rect——矩形，如条形图和时间线。

rule—规则是线段，通常用于轴刻度线和网格线。

shape-一个特殊的路径标记的变体，以更快地绘制地图。

symbol-绘制符号，包括圆形、正方形和其他形状。

text -文本标签可配置字体、对齐方式和角度。

trail -可以根据底层数据改变大小的行。

具体讲解如下链接所示：

<https://vega.github.io/vega/docs/marks/>

**Signals**

信号是参数化可视化的动态变量，可以驱动交互行为。信号可以在整个Vega规范中使用，例如定义标记属性或数据转换参数。

信号值是反应性的:它们可以更新以响应输入事件流、外部API调用或上游信号的更改。事件流捕获和序列输入事件，如mousedown或touchmove。当事件发生时，与相关事件处理程序相关联的信号将按其规范顺序重新计算。然后，更新的信号值传播到规范的其余部分，并自动重新呈现可视化。

信号定义及其在规范中其余部分的使用看起来像这样:

{

**"signals"**: [

{

**"name"**: "indexDate",

**"description"**: "A date value that updates in response to mousemove.",

**"update"**: "datetime(2005, 0, 1)",

**"on"**: [{**"events"**: "mousemove", **"update"**: "invert('xscale', x())"}]

}

],

**"data"**: [

{ **"name"**: "stocks", ... },

{

**"name"**: "index",

**"source"**: "stocks",

**"transform"**: [

{

**"type"**: "filter",

**"expr"**: "month(datum.date) === month(indexDate)"

}

]

}

],

**"scales"**: [

{ **"name"**: "x", **"type"**: "time", ... }

],

**"marks"**: [

{

**"type"**: "rule",

**"encode"**: {

**"update"**: {

**"x"**: {**"scale"**: "x", **"signal"**: "indexDate"}

}

}

}

]}

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| name | [String](https://vega.github.io/vega/docs/types/" \l "String) | 必需的。信号的唯一名称。信号名应该是有效的JavaScript标识符:它们应该只包含字母数字字符(或" $ "或" \_ ")，并且不能以数字开头。不能用作信号名称的保留关键字是“datum”、“event”、“item”和“parent”。 |
| bind | [Bind](https://vega.github.io/vega/docs/signals/" \l "bind) | 将信号绑定到外部输入元素，如滑块、选择列表或单选按钮组。 |
| description | [String](https://vega.github.io/vega/docs/types/" \l "String) | 信号的文本描述，对内联文档很有用。 |
| on | [Handler](https://vega.github.io/vega/docs/signals/" \l "handlers)[ ] | 一组事件流处理程序，用于更新响应输入事件的信号值。 |
| init | [Expression](https://vega.github.io/vega/docs/expressions) | ≥4.4信号值的初始化表达式。这个表达式将被调用一次且只调用一次。init和update参数是互斥的，不能同时使用。 |
| update | [Expression](https://vega.github.io/vega/docs/expressions) | 信号值的更新表达式。该表达式可能包括其他信号，在这种情况下，信号将自动更新以响应上游信号的变化，只要react属性不为false。init和update参数是互斥的，不能同时使用。 |
| react | [Boolean](https://vega.github.io/vega/docs/types/" \l "Boolean) | 一个布尔标志(默认为true)，指示在任何上游信号依赖项更新时是否应该自动重新计算更新表达式。如果为false，更新表达式将不会注册任何对其他信号的依赖，即使是初始化。 |
| value | [Any](https://vega.github.io/vega/docs/types/" \l "Any) | 信号的初始值(默认未定义)。该值在init或update表达式之前赋值。 |

**Event Streams**

事件流是建模用户输入以实现动态、交互式可视化的主要方法。事件流捕获一系列输入事件，如鼠标点击、触摸移动、计时器滴答或信号更新。当匹配流定义的事件发生时，它们将导致任何相应的信号事件处理程序进行计算，从而可能更新信号值。

{

**"name"**: "signalName",

**"on"**: [

{

**"events"**: <<event-stream-definition>>,

**"update"**: ...

}

]}

事件流定义可以通过多种方式指定:

事件流对象指示要捕获哪些事件。

事件流选择器字符串为事件流对象提供了方便的简写。例如，“rect:mouseover”，“click[event]”。shiftKey]”或“timer”{1000}。

信号引用捕获信号更新。例如，{"signal": "name"}。

缩放引用捕获缩放更新。例如:{"scale": "xscale"}。

捕获的多个事件流对象、信号引用或缩放引用的数组。

具体讲解如下链接所示：

<https://vega.github.io/vega/docs/event-streams/>

**Expressions**

为了启用自定义计算，Vega包含了自己的表达式语言，用于编写基本公式。例如，过滤器和公式转换使用这些表达式修改数据，并在信号定义中计算响应用户输入的更新值。

表达式语言是JavaScript的一个受限子集。支持所有基本的算术、逻辑和属性访问表达式，以及布尔、数字、字符串、对象({})和数组([])字面量。三元操作符(ifTest ?thenValue: elseValue)和一个特殊的if(test, thenValue, elseValue)函数。

为了保持表达式语言的简单、安全并避免不必要的副作用，以下元素是不允许的:赋值操作符(=、+=等)、前置/后置更新(++)、new表达式和大多数控制流语句(for、while、switch等)。此外，不允许调用涉及嵌套属性(foo.bar())的函数。相反，表达式语言支持在顶级作用域中定义的函数集合。

本页记录了表达式语言。如果您对实现方面感兴趣，那么vega-expression模块提供了表达式语言的大部分内容——包括解析、代码生成以及一些常量和函数定义。

具体讲解如下链接所示：

<https://vega.github.io/vega/docs/expressions/>

**Layout**

布局在网格中定位组标记的集合，简化了小多个和协调的多个视图显示的组成。当应用于规范的顶层或组标记内时，所有直接的子组标记将被收集并根据布局规范定位。布局引擎支持流布局以及列、行和网格对齐布局。

布局引擎还支持包含行和列的页眉和页脚单元格，以及行标题和列标题单元格。为了指示页眉、页脚和标题，这些组的规范必须包含一个角色属性，该属性设置为column-header、column-footer、column-title、行-header、行-footer或行-title之一。页眉、页脚或标题元素的数量应该与表中的行数或列数匹配。如果元素较少，一些单元格将被保留为空。如果元素太多，则会忽略其他元素，并记录一个警告。

布局中组的顺序取决于规范顺序(跨组标记定义)和内部标记顺序(对于单个组标记定义中的多个组实例)。组标记定义在规范中出现的顺序决定了它们在布局中的顺序。在一个具有多个组实例的组标记规范中，组项的内部顺序决定了它们的呈现顺序和它们在布局中的顺序。内部顺序可以使用sort指令修改。

具体讲解如下链接所示：

<https://vega.github.io/vega/docs/layout/>

**Types**

Reference documentation for common parameter **types** expected by Vega specification properties.

## Parameter Type Reference

* [Any](https://vega.github.io/vega/docs/types/" \l "Any)
* [Array](https://vega.github.io/vega/docs/types/" \l "Array)
* [Boolean](https://vega.github.io/vega/docs/types/" \l "Boolean)
* [Color](https://vega.github.io/vega/docs/types/" \l "Color)
* [Date](https://vega.github.io/vega/docs/types/" \l "Date)
* [Gradient](https://vega.github.io/vega/docs/types/" \l "Gradient)
* [Number](https://vega.github.io/vega/docs/types/" \l "Number)
* [Object](https://vega.github.io/vega/docs/types/" \l "Object)
* [String](https://vega.github.io/vega/docs/types/" \l "String)
* [URL](https://vega.github.io/vega/docs/types/" \l "URL)
* [Data](https://vega.github.io/vega/docs/types/" \l "Data)
* [Field](https://vega.github.io/vega/docs/types/" \l "Field)
* [Signal](https://vega.github.io/vega/docs/types/" \l "Signal)
* [Compare](https://vega.github.io/vega/docs/types/" \l "Compare)
* [Expr](https://vega.github.io/vega/docs/types/" \l "Expr)
* [Value](https://vega.github.io/vega/docs/types/" \l "Value)
* [ColorValue](https://vega.github.io/vega/docs/types/" \l "ColorValue)
* [FieldValue](https://vega.github.io/vega/docs/types/" \l "FieldValue)
* [GradientValue](https://vega.github.io/vega/docs/types/" \l "GradientValue)
* [TimeMultiFormat](https://vega.github.io/vega/docs/types/" \l "TimeMultiFormat)

具体讲解如下链接所示：

<https://vega.github.io/vega/docs/types/>

1. 项目为例讲解

|  |  |
| --- | --- |
| 示例代码 | 说明 |
| {  "$schema": "https://vega.github.io/schema/vega-lite/v4.json",  "title": "各用户的使用量统计",  "data": {  "url": {  "index": "hpc\_job",  "body": {  "size": 10000,  "\_source": ["username","start\_time","end\_time","processors"]  }  },  "format": {"property": "hits.hits"}  },  "transform": [  {  "calculate": "datum.\_source.username",  "as":"user"  },  {  "calculate":  "toNumber(datetime(datum.\_source['start\_time'])) >= toNumber(datetime(2020,2,14)) && toNumber(datetime(datum.\_source['end\_time'])) <= toNumber(datetime(2021,3,14)) && toNumber(datetime(datum.\_source['end\_time'])) > toNumber(datetime(2020,2,14)) ? toNumber(datetime(datum.\_source['end\_time']) - datetime(datum.\_source['start\_time']))\*datum.\_source.processors : 0",  "as":"usage1"  },  {  "calculate":  "toNumber(datetime(datum.\_source['end\_time'])) > toNumber(datetime(2021,3,14)) && toNumber(datetime(datum.\_source['start\_time'])) < toNumber(datetime(2020,2,14)) ? (toNumber(datetime(2021,3,14))-toNumber(datetime(2020,2,14)))\*datum.\_source.processors : 0",  "as":"usage2"  },  {  "calculate":  "toNumber(datetime(datum.\_source['start\_time'])) >= toNumber(datetime(2020,2,14)) && toNumber(datetime(datum.\_source['start\_time'])) < toNumber(datetime(2021,3,14)) && toNumber(datetime(datum.\_source['end\_time'])) > toNumber(datetime(2021,3,14)) ? (toNumber(datetime(2021,3,14))-toDate(datum.\_source['start\_time']))\*datum.\_source.processors : 0",  "as":"usage3"  },  {  "calculate":  " toNumber(toDate(datum.\_source['end\_time'])) > toNumber(datetime(2020,2,14)) && toNumber(toDate(datum.\_source['end\_time'])) <= toNumber(datetime(2021,3,14)) && toNumber(toDate(datum.\_source['start\_time'])) < toNumber(datetime(2020,2,14)) ? (toNumber(toDate(datum.\_source['end\_time']) - toNumber(datetime(2020,2,14))))\*datum.\_source.processors : 0",  "as":"usage4"  },  {  "calculate":  "datum.usage1+datum.usage2+datum.usage3+datum.usage4",  "as":"usage5"  },  {  "filter":  toNumber(datetime(datum.\_source['end\_time'])) >= toNumber(datetime(datum.\_source['start\_time']))  }  ],  "mark": "bar",  "encoding": {  "x": {"field": "user", "type": "nominal","axis": {"title": "用户"}},  "y": {  "field": "usage5",  "type": "quantitative",  "axis": {"title": "总使用量"}  }  }  } | 在这个例子中，首先在schema中我们使用vega-lite格式中的第四版本，即v4.json格式。  整个图形的title名称为“各用户的使用量统计”。  在数据集中，我们使用url的方式，通过索引“index”获取指定的数据表“hpc-job”，并指定获取的数据条数为10000条，“-source”表示获取的数据表中的数据项分别是什么。“format”表示我们获取的数据来自哪个字段的哪个数据。  “transform”表示我们需要对获取的字段进行一些转换操作（在transform中引入当前字段的时候需要在该字段前加上datum前缀）。  “calculate”表示对当前获取的字段进行一些计算操作和取值。其中该项目中由于每个计算的条件不同，而filter字段只能针对总的transform限定，因此我们选择在每个“calculate”之后采用三目运算符来进行判断“x>y?x;y”表示如果x大于y的话，取x值，否则取y值。”as“表示对计算的结果进行存储。  ”mark“：Bar表示我要显示的图为柱状图。同时也有line图（线图），pie图（饼图），point图（散点图）等，根据需要选择合适的类型即可。  ”encoding“表示我需要怎么进行展示，x和y分别表示图形的xhey轴，field表示显示的字段（之前存储的），type有三种，分别是quantitative（数字型），nominal（字符串型），以及temporal（日期型），我们根据字段选择合适的类型，axis中我们也可以定义title显示横坐标和纵坐标分别代表什么（起名）。 |

具体图例显示如下所示：

