VEGA

# 基本概念

基于D3之上的高层可视化指定语言（a higher-level visualization specification language on top of D3）

## 设计目标

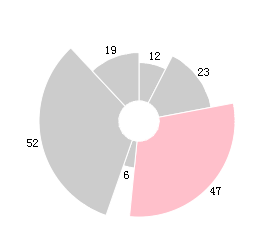
* 快速自定义设计
* 重用和分享
* 编程式生成可视化
* 效率和可移植性（并非指WEB本身的跨平台性，这种描述式语言本身可以不依赖于JS）

## 版本

目前存在两个版本，稳定版1.5(stable)和测试版2.0(dev)，测试版中增加了自定义交互的设计，以及一些细节方面的改进。2.0的改变其实可以看作其对自身缺点的认知，可以仔细研究一下，文中会对2.0的改进进行标识。

# 内容分析

语言本身是JSON的结构，所以表现为嵌套结构，同时需要为不同属性设定值。样例：



|  |
| --- |
| {  **图的基本信息**  "name": "arc",  "width": 400,  "height": 400,  **定义数据**  "data": [  {  "name": "table",  "values": [12, 23, 47, 6, 52, 19],  **数据变换**  "transform": [  {"type": "pie", "value": "data"}  ]  }  **定义Scale**  ],  "scales": [  {  "name": "r",  "type": "sqrt",  "domain": {"data": "table", "field": "data"},  "range": [20, 100]  }  **定义图元**  ],  "marks": [  {  **图元类型、数据来源、基本属性绑定**  "type": "arc",  "from": {"data": "table"},  "properties": {  "enter": {  "x": {"group": "width", "mult": 0.5},  "y": {"group": "height", "mult": 0.5},  "startAngle": {"field": "startAngle"},  "endAngle": {"field": "endAngle"},  "innerRadius": {"value": 20},  "outerRadius": {"scale": "r"},  "stroke": {"value": "#fff"}  },  **简单的交互设定**  "update": {  "fill": {"value": "#ccc"}  },  "hover": {  "fill": {"value": "pink"}  }  }  },  {  "type": "text",  "from": {"data": "table"},  "properties": {  "enter": {  "x": {"group": "width", "mult": 0.5},  "y": {"group": "height", "mult": 0.5},  "radius": {"scale": "r", "offset": 8},  "theta": {"field": "midAngle"},  "fill": {"value": "#000"},  "align": {"value": "center"},  "baseline": {"value": "middle"},  "text": {"field": "data"}  }  }  }  ]  } |

|  |  |
| --- | --- |
| **特点** | **与VisComposer的对比** |
| 除了图的基本信息部分，其他内容（数据、变换、图元等）可以在mark后的大括号内继续出现形成嵌套。 | 同样是树形结构，事实上由于是基于D3所以基本上注定了树形结构。 |
| 数据绑定可以使用变量名（data、data.x等）、简单的变换（mult和offset属性）或者通过expr属性绑定一个算式。 | 非常类似于VisComposer中的直接绑定、Modifier和Expression Shader。Modifier能提供的内容相对简单的乘和加更多一些。 |
| 数据变换可以出现在数据部分或者图元绑定部分，提供基础的几种变换，或者选择输入自定义算式（expr）。 | 相当于Transformation Module，不过结构简单得多，基本上就是一个简单的算式或者预定义的变换（如一些布局） |
| layout通过transform部分定义，然后将计算结果直接用于图元的绑定。   |  | | --- | | transform": [  {"type": "pie", "value": "data"}  ] | | 这里的type属性实际上相当于d3.layout的成员。这个方法与VisComposer中使用的非常类似。 |
| 有一些简单且固定的交互绑定。新版添加了更加复杂的交互设计，详见下文。 |  |

**新版（2.0）添加的交互设计**

信号（signal）的定义。交互的设置基于信号的定义和响应

|  |
| --- |
| "signals": [  {  "name": "clickedPt",  "init": 0,  "verbose": true,  "streams": [{"type": "click", "expr": "datum.\_id"}]  },  ] |

触发后进行“判定”，实际就是进行一些数据的标记（如是否被选中）、一些属性的记录（如brush的范围）

|  |
| --- |
| "predicates": [  {  "name": "isSelected",  "type": "in",  "item": {"arg": "id"},  "data": "selectedPts",  "field": "id"  }  ], |

数据绑定时可以根据这些标记进行不同的绑定，以表达出交互结果

|  |
| --- |
| "fill": {  "rule": [  {  "predicate": {  "name": "isSelected",  "id": {"field": "\_id"}  },  "scale": "c",  "field": "species"  },  {"value": "grey"}  ]  } |

这部分可以参考。实际上我之前思考交互设计的时候也在考虑信号的机制，详细内容还需要进一步考虑。

总体来说：

* 结构简单明了，但不能完全脱离编程式设计的高初始学习成本。例如transform（尤其是作为layout存在的transform）的输出，在之后需要绑定时无法直观的看出哪个变量代表这些生成出来的量（可能就是a.data甚至就是a这样意义不明的名称，在**2.0**中有所改进，例如统一由layout\_作为前缀）。其原因在于没有GUI所以缺乏提示信息，同时又是在编程语言之上包装了一层，在某些方面甚至不如直接编程直观。这里是VisComposer可以有的优势所在。
* 值得注意的一个细节是引号。在做VisComposer的时候也注意到的一个问题就是引号是否要加，对于用户来说很难判断，但加或者不加是完全不懂的意思。VEGA中大部分内容都带有引号也使得看起来有些杂乱。

**一些其他特性**

* 动画。提供了简单的基于数值插值的渐变效果。但样例中并未提供相应展示，效果未知。
* 非浏览器端绘制。VEGA可以通过Node.js进行后端的并行绘制。通过命令行进行绘制的调用。
* 可以使用不包含DOM结构的canvas进行绘制并提供交互事件响应，这也是VEGA与D3的不同之处。

# 实现细节

* 部分地方调用了一些d3函数，但实际d3并没有处于基础的位置，仅仅使用了里面的一些实用函数（range、scale、一些layout等），这一点与VisComposer也很像
* 绘制方面没有什么特别的地方，就是记录下所有的mark然后一个个按参数绘制上去。没有找到和node.js关联的部分
* 交互的事件响应写的非常精巧，可以考虑参考使用
* 值得一提的是expression的部分，VEGA是实现了一个简单的语法解析器的，可以考虑拿来使用
* 总体来说代码非常轻量级，简洁直观，但细节部分比较完善

# 总结

总体来说和VisComposer非常相近，甚至可以看作是去除了GUI和一些可编程部分的简化版VisComposer，外加一些交互的设计。总体虽然并不复杂，但是很多部分设计的都比较合理和精致，可以作为参考。