# Echarts

npm install echarts --save

通过 npm 安装的 echarts 会出现在 myProject/node\_modules

使用 ES Module：

import \* as echarts from 'echarts';

使用 CommonJS：

var echarts = require('echarts')

## 使用 echarts 提供的构建脚本自定义构建

想要引入的文件小一些。

可是按教程做失败了

## echarts 实例

每个 echarts 实例独占一个 DOM 节点。

每个实例可以有很多张图

option 描述实例

var dom = document.getElementById('dom-id');

var chart = echarts.init(dom);

创建 echarts 实例

## 系列series

系列（series）是指：一组数值和他们映射成的图。所以，一个系列包含的要素至少有：一组数值、图表类型（series.type）、以及其他的关于这些数据如何映射成图的参数。

系列类型series.type就是图表类型。series.type至少有：line（折线图）、bar（柱状图）、pie（饼图）、scatter（散点图）、graph（关系图）、tree（树图）

series: [

{

name: 'pm2.5',

type: 'scatter',

coordinateSystem: 'geo',

data: ...,

label: {

normal: {

show: false

},

emphasis: {

show: false

}

},

itemStyle: {

emphasis: {

borderColor: '#fff',

borderWidth: 1

}

}

}

]

* name 系列名
* type 图表类型
* data 数据，也可以写在外边
* label 标签
* symbolSize: 12 散点图特有，散点大小
* itemStyle 散点样式

## 组件component

在系列之上，echarts 中各种内容，被抽象为“组件”。

option 每个属性是一类组件。

echarts 中至少有这些组件：xAxis、yAxis、grid、angleAxis（极坐标系角度轴）、radiusAxis（极坐标系半径轴）、polar（极坐标系底板）、geo（地理坐标系）、dataZoom（数据区缩放组件）、visualMap（视觉映射组件）、tooltip（提示框组件）、toolbox（工具栏组件）...

系列series也是一种组件，是专门绘制“图”的组件。

## option

option 表述了：数据、数据如何映射成图形、交互行为。

option属性：

backgroundColor: '#404a59',

背景颜色

title: {

text: '全国主要城市空气质量',

subtext: 'data from PM25.in',

sublink: 'http://www.pm25.in',

left: 'center',

textStyle: {

color: '#fff'

}

},

* 标题
* 子标题
* 链接
* 标题位置
* 样式

tooltip: {

trigger: 'item'

}

提示框

legend: {

orient: 'vertical',

top: 'bottom',

left: 'right',

data:['pm2.5'],

textStyle: {

color: '#fff'

}

}

图例，说明

visualMap: { }

见[VisualMap组件](#_VisualMap组件)

toolbox: {

show : true,

orient : 'vertical',

left: 'right',

top: 'center',

feature : {

mark : {show: true},

dataView : {show: true, readOnly: false},

restore : {show: true},

saveAsImage : {show: true}

}

}

工具栏

* 展示数据视图
* 还原
* 保存为图片

var option = {

xAxis: [

{type: 'category', ...},

{type: 'category', ...},

{type: 'value', ...}

],

yAxis: [{...}, {...}],

series: [

{type: 'line', data: [['AA', 332], ['CC', 124], ['FF', 412], ... ]},

{type: 'line', data: [2231, 1234, 552, ... ]},

{type: 'line', data: [[4, 51], [8, 12], ... ]}

}]

};

chart.setOption(option);

series里的data 是本系列的数据。而另一种描述方式，系列数据从 dataset 中取：

var option = {

dataset: {

source: [

[121, 'XX', 442, 43.11],

[663, 'ZZ', 311, 91.14],

[913, 'ZZ', 312, 92.12],

...

]

},

xAxis: {},

yAxis: {},

series: [

// 数据从 dataset 中取，encode 中的数值是 dataset.source 的维度 index （即第几列）

{type: 'bar', encode: {x: 1, y: 0}},

{type: 'bar', encode: {x: 1, y: 2}},

{type: 'scatter', encode: {x: 1, y: 3}},

...

]

};

## 定位

类CSS的绝对定位

* 绝对数值，如top:52，表示距离Echarts容器顶边接52像素
* 百分比，如left:'30%'，基于Echarts容器宽高

var option = {

grid: {

left: 130,

right: '30%‘，

height: '40%',

bottom: 110

}

}

## 坐标系

直角坐标系中，包括有 xAxis（X轴）、yAxis（Y轴）、grid（直角坐标系底板）三种组件

var option = {

xAxis: {},

yAxis: {},

series: {

type :'scatter',

data: [

[13, 44],

[51, 51],

[51, 32],

[67, 19]

]

}

}

声明了 xAxis、yAxis 和一个 scatter散点图系列，echarts 暗自为他们创建了 grid

grid:[

{top: 40, bottom: '58%'},

{top: '58%', bottom: 40}

],

xAxis: [

{type: 'category', gridIndex: 0},

{type: 'category', gridIndex: 1},

],

yAxis: [

{type: 'value', gridIndex: 0},

{type: 'value', gridIndex: 1},

],

series: [{

type: 'line',

xAxisIndex: 0,

yAxisIndex: 0

}, {

type: 'line',

xAxisIndex: 1,

yAxisIndex: 1

}, {

type: 'bar',

xAxisIndex: 1,

yAxisIndex: 1

}]

一个grid里有两张图，第二张图里有柱状和折线两个表

xAxis属性可以是对象，也可以是由对象构成的数组

对象有属性

* type 可以是value或category
* gridIndex
* data 一个数组

## 个性化样式

饼状图

myChart.setOption({

series : [

{

name: '访问来源',

type: 'pie',

radius: '55%',

data:[

{value:235, name:'视频广告'},

{value:274, name:'联盟广告'},

{value:310, name:'邮件营销'},

{value:335, name:'直接访问'},

{value:400, name:'搜索引擎'}

]

}

]

})

radius相当于大小

饼图支持通过设置 roseType 显示成南丁格尔图。

roseType: 'angle'

南丁格尔图会通过半径表示数据的大小。

阴影、透明度、颜色、边框颜色、边框宽度等，一般都会在系列的 itemStyle 里设置。

* 设置阴影

itemStyle: {

// 阴影的大小

shadowBlur: 200,

// 阴影水平方向上的偏移

shadowOffsetX: 0,

// 阴影垂直方向上的偏移

shadowOffsetY: 0,

// 阴影颜色

shadowColor: 'rgba(0, 0, 0, 0.5)'

}

* 扇形颜色

itemStyle: {

color: '#c23531',

}

深色背景和浅色标签

现在我们需要把整个主题改成深色主题，这就需要改背景色和文本颜色。

itemStyle的emphasis属性设置鼠标 hover 时的高亮样式。

itemStyle: {

emphasis: {

shadowBlur: 200,

shadowColor: 'rgba(0, 0, 0, 0.5)'

}

}

* 背景色

背景色是全局的，直接在 option 下设置 backgroundColor

setOption({

backgroundColor: '#2c343c'

})

* 文本样式

文本的样式可以设置全局的 textStyle

setOption({

textStyle: {

color: 'rgba(255, 255, 255, 0.3)'

}

})

也可以每个系列分别设置

series: [

{

label: {

textStyle: {

color: 'rgba(255, 255, 255, 0.3)'

}

}

}

]

* 饼图的视觉引导线颜色

饼图的话还可以设置标签的视觉引导线的颜色

labelLine: {

lineStyle: {

color: 'rgba(255, 255, 255, 0.3)'

}

}

option = {

backgroundColor: '#2c343c',

series : [

{

name: '访问来源',

type: 'pie',

radius: '55%',

data:[

{value:235, name:'视频广告'},

{value:274, name:'联盟广告'},

{value:310, name:'邮件营销'},

{value:335, name:'直接访问'},

{value:400, name:'搜索引擎'}

],

roseType: 'angle',

label: {

normal: {

textStyle: {

color: 'rgba(255, 255, 255, 0.3)'

}

}

},

labelLine: {

normal: {

lineStyle: {

color: 'rgba(255, 255, 255, 0.3)'

}

}

},

itemStyle: {

normal: {

color: '#c23531',

shadowBlur: 200,

shadowColor: 'rgba(0, 0, 0, 0.5)'

}

}

}

]

};

教程示例代码中每一个label、labelLine、itemStyle中都要加一个normal，但其实不加也行。normal对应emphasis，指的是正常情况下的样式。

* 颜色主题（Theme）

var chart = echarts.init(dom, 'light');

或者

var chart = echarts.init(dom, 'dark');

其他的主题，没有内置在 ECharts 中，需要自己加载。这些主题可以在主题编辑器里访问到。也可以使用这个主题编辑器，自己编辑主题。

* 调色盘

调色盘，可以在 option 中设置全局调色盘，也可以设置系列自己专属的调色盘

它给定一组颜色，图形、系列会自动从其中选择颜色

option = {

// 全局调色盘。

color: ['#c23531','#2f4554', '#61a0a8', '#d48265', '#91c7ae','#749f83', '#ca8622', '#bda29a','#6e7074', '#546570', '#c4ccd3'],

series: [{

type: 'bar',

color: ['#dd6b66', '#759aa0', '#e69d87', '#8dc1a9', '#ea7e53', '#eedd78', '#73a373', '#73b9bc', '#7289ab', '#91ca8c','#f49f42'],

}, {

type: 'pie',

color: ['#37A2DA', '#32C5E9', '#67E0E3', '#9FE6B8', '#FFDB5C','#ff9f7f', '#fb7293', '#E062AE', '#E690D1', '#e7bcf3', '#9d96f5', '#8378EA', '#96BFFF'],

}]

}

## 异步加载和更新数据

var myChart = echarts.init(document.getElementById('main'));

$.get('data.json').done(function (data) {

myChart.setOption({

series: [{

name: '销量',

type: 'bar',

data: [5, 20, 36, 10, 10, 20]

}]

});

});

一种就像这样，先init，等数据拿回来，再传参

var myChart = echarts.init(document.getElementById('main'));

// 显示标题，图例和空的坐标轴

myChart.setOption({

title: {

text: '异步数据加载示例'

},

xAxis: {

data: []

},

yAxis: {},

series: [{

name: '销量',

type: 'bar',

data: []

}]

});

// 异步加载数据

$.get('data.json').done(function (data) {

// 填入数据

myChart.setOption({

xAxis: {

data: data.categories

},

series: [{

// 根据名字对应到相应的系列

name: '销量',

data: data.data

}]

});

});

一种就像这样，先init并setOption，显示空的坐标轴，然后接受到数据以后再来一次setOption,传入数据

注意，series里有一个属性name，可以通过name区别输入数据的图是哪个。

所以说，后setOpion的把前边的都覆盖了，应该是。

## loading 动画

myChart.showLoading();

$.get('data.json').done(function (data) {

myChart.hideLoading();

myChart.setOption(...);

});

仔细体会啊，仔细体会

先有loading动画，然后接收到数据后停止动画，显示图表

## dataset

option = {

dataset: {

source: [

['product', '2015', '2016', '2017'],

['Matcha Latte', 43.3, 85.8, 93.7],

['Milk Tea', 83.1, 73.4, 55.1],

['Cheese Cocoa', 86.4, 65.2, 82.5],

['Walnut Brownie', 72.4, 53.9, 39.1]

]

},

xAxis: {type: 'category'},

yAxis: {},

series: [

{type: 'bar'},

{type: 'bar'},

{type: 'bar'}

]

}

声明一个 X 轴，类目轴（category）。默认情况下，类目轴对应到 dataset 第一列。

声明一个 Y 轴，数值轴。默认就是数值轴

声明多个 bar 系列，默认情况下，每个系列会自动对应到 dataset 的每一列。

也就是说，默认情况下，m行n列的矩阵，[aij]，1≤i≤m, 1≤j≤n

* a11-am1 是x轴的类目，柱状图的横坐标。可以是数字，可以是字符
* a12-a1n 此坐标系有n-1个系列。a12-a1n就相当于每个系列的x轴横坐标，或者每个系列的名字？
* ai2-ain, i!=1 每个类目内部的数据。n=2时，就是普通的柱状图
* a2j-amj, 2≤j≤n 当j=2时，就是每个类目第一个柱子的数据，也就是第一个series的数据。以此类推。即，系列被安放到 dataset 的列上面

dataset: {

// 这里指定了维度名的顺序，从而可以利用默认的维度到坐标轴的映射。

dimensions: ['product', '2015', '2016', '2017'],

source: [

{product: 'Matcha Latte', '2015': 43.3, '2016': 85.8, '2017': 93.7},

{product: 'Milk Tea', '2015': 83.1, '2016': 73.4, '2017': 55.1},

{product: 'Cheese Cocoa', '2015': 86.4, '2016': 65.2, '2017': 82.5},

{product: 'Walnut Brownie', '2015': 72.4, '2016': 53.9, '2017': 39.1}

]

},

数据到图形的映射

制作数据可视化图表的逻辑是：基于数据，在配置项中指定如何映射到图形。

可以进行这些映射：

* 使用seriesLayoutBy指定 dataset 的列还是行映射为图形系列。
* 使用encode指定维度映射的规则：如何从 dataset 的维度映射到坐标轴（如 X、Y 轴）、提示框（tooltip）、标签（label）、图形元素大小颜色等（visualMap）。这件事可以使用 series.encode 属性，以及 visualMap 组件来配置。
* 使用默认映射规则

在坐标系中（如直角坐标系、极坐标系等），如果有类目轴（axis.type 为 'category'），则将第一列（行）映射到这个轴上，后续每一列（行）对应一个系列。

如果没有类目轴，假如坐标系有两个轴（例如直角坐标系的 X Y 轴），则每两列对应一个系列，这两列分别映射到这两个轴上。

如果没有坐标系（如饼图），取第一列（行）为名字，第二列（行）为数值（如果只有一列，则取第一列为数值）。

使用 seriesLayoutBy 配置项，改变图表对于行列的理解。seriesLayoutBy 可取值：

* 'column': 默认值。系列被安放到 dataset 的列上面。
* 'row': 系列被安放到 dataset 的行上面。

dataset: {

source: [

['product', '2012', '2013', '2014', '2015'],

['Matcha Latte', 41.1, 30.4, 65.1, 53.3],

['Milk Tea', 86.5, 92.1, 85.7, 83.1],

['Cheese Cocoa', 24.1, 67.2, 79.5, 86.4]

]

},

series: [

{type: 'bar', seriesLayoutBy: 'row'},

{type: 'bar', seriesLayoutBy: 'row'},

{type: 'bar', seriesLayoutBy: 'row'},

]

在此坐标系中，每个系列对应到 dataset 的每一行。

即dataset.source有m-1个系列

**维度**dimension

常用图表所描述的数据大部分是“二维表”结构。当series对应到“列”的时候，每一列就称为一个维度，而每一行称为数据项（item）。

反之，如果series对应到表行，那么每一行就是“维度”，每一列就是数据项。

维度可以有单独的名字，便于在图表中显示。维度名dimension name可以在定义在 dataset 的第一行（列）。

dataset.source 中第一行（列）到底包含不包含维度名，ECharts 默认会自动探测。也可以设置 dataset.sourceHeader: true 显式声明第一行（列）就是维度名，或者 dataset.sourceHeader: false 表明第一行（列）开始就直接是数据。

自动检测的机制可能是判断是不是string

维度可以使用单独的 dataset.dimensions 或 series.dimensions定义

var option1 = {

dataset: {

dimensions: [

{name: 'score'},

'amount',

{name: 'product', type: 'ordinal'} //'product'是数据中的a11

],

source: [...]

}

};

以上是三种维度声明方法。也同时声明了3个维度

* dimensions 是一个数组。数组元素是对象。对象包含一个属性name表示维度名称
* 对象可以简写为string
* 对象可以有type属性指定维度类型
* 可以设置为 null 表示不想设置维度名

var option2 = {

dataset: {

source: [...]

},

series: {

type: 'line',

dimensions: [

null,

'amount',

{name: 'product', type: 'ordinal'}

]

}

};

* 在系列中设置的 dimensions 会更优先采纳。
* 大多数情况下并不需要去设置维度类型，因为会自动判断。但是如果因为数据为空之类原因导致判断不足够准确时，可以手动设置维度类型。

维度类型可以取值：

* 'number': 默认，表示普通数据
* 'ordinal': 对于类目、文本这些 string 类型的数据，如果需要能在数轴上使用，须是 'ordinal' 类型。ECharts 默认会自动判断这个类型。但是自动判断也是不可能很完备的，所以使用者也可以手动强制指定。
* 'time': 表示时间数据。时间类型的支持参见 data。
* 'float': 如果设置成 'float'，在存储时候会使用 TypedArray，对性能优化有好处。
* 'int': 如果设置成 'int'，在存储时候会使用 TypedArray，对性能优化有好处。

使用**encode**做数据到图形的映射

var option = {

dataset: {

source: [

['score', 'amount', 'product'],

[89.3, 58212, 'Matcha Latte'],

[57.1, 78254, 'Milk Tea'],

[74.4, 41032, 'Cheese Cocoa'],

[50.1, 12755, 'Cheese Brownie'],

[89.7, 20145, 'Matcha Cocoa'],

[68.1, 79146, 'Tea'],

[19.6, 91852, 'Orange Juice'],

[10.6, 101852, 'Lemon Juice'],

[32.7, 20112, 'Walnut Brownie']

]

},

xAxis: {},

yAxis: {type: 'category'},

series: [

{

type: 'bar',

encode: {

// 将 "amount" 列映射到 X 轴。

x: 'amount',

// 将 "product" 列映射到 Y 轴。

y: 'product'

}

}

]

};

encode属性名是坐标系、标签等特定名称，如 'x', 'y', 'tooltip' 等

属性值是data.source中的维度名，或者维度的序号（从 0 开始计数），可以使用数组指定一个或多个维度，但是不知道有什么效果

encode: {

tooltip: ['product', 'score']

seriesName: [1, 3],

itemId: 2,

itemName: 3

}

tooltip 提示框，显示详细信息

seriesName

itemId

itemName 以上属性还不知道是干什么的

直角坐标系（grid/cartesian）特有的属性：

encode: {

// 把 “维度1”、“维度5”、“名为 score 的维度” 映射到 X 轴：为什么能映射那么多？

x: [1, 5, 'score'],

y: 0

}

单轴（singleAxis）特有的属性：

encode: {

single: 3

}

极坐标系（polar）特有的属性：

encode: {

radius: 3,

angle: 2

}

地理坐标系（geo）特有的属性：

encode: {

lng: 3,

lat: 2

}

对于一些没有坐标系的图表，例如饼图、漏斗图等，可以是：

encode: {

value: 3

}

## VisualMap组件

visualMap 组件定义了把数据的哪个维度映射到什么视觉元素上。

visualMap组件类型通过 visualMap.type 区分。

option = {

visualMap: [ // 可以同时定义多个 visualMap 组件

{

type: 'continuous', // 定义为连续型 visualMap

},

{

type: 'piecewise', // 定义为分段型 visualMap

}

]

}

type: 'continuous' 连续型 visualMap

type: 'piecewise' 分段型 visualMap

连续型视觉映射组件 visualMapContinuous

option = {

visualMap: {

type: 'continuous',

min: 800,

max: 50000,

text:['High','Low'],

realtime: false,

calculable : true,

color: ['orangered','yellow','lightskyblue']

}

});

* min

指定 visualMapContinuous 组件的允许的最小值。'min' 必须用户指定。

* max

指定 visualMapContinuous 组件的允许的最大值。'max' 必须用户指定。[visualMap.min, visualMax.max] 形成了视觉映射的『定义域』。

* range

Array类型

指定手柄对应数值的位置。range 应在 min max 范围内

不设置 range，则 range 默认为 [min, max]

* calculable

是否显示拖拽用的手柄（手柄能拖拽调整选中范围）。

* realtime

拖拽时，是否实时更新。

如果为ture则拖拽手柄过程中实时更新图表视图。

如果为false则拖拽结束时，才更新视图。

* itemWidth

长条宽度

* itemHeight

长条高度

* align = 'auto'

指定组件中手柄和文字的摆放位置，可选值

* 'auto' 自动决定。
* 'left' 手柄和label在右，orient 为 horizontal 时有效。
* 'right' 手柄和label在左，orient 为 horizontal 时有效。
* 'top' 手柄和label在下，orient 为 vertical 时有效。
* 'bottom' 手柄和label在上，orient 为 vertical 时有效。
* show : true

是否显示 visualMap-continuous 组件。如果设置为 false，不会显示，但是数据映射的功能还存在。

* dimension:1

指定用数据的『哪个维度』，映射到视觉元素上。『数据』即 series.data。

默认取 data 中最后一个维度。

* hoverLink : true

打开 hoverLink时，鼠标悬浮到 visualMap 组件上，鼠标位置对应的数值在图表中对应的图形元素，会高亮。

* inRange

定义在选中范围中的视觉元素样式

可选的视觉元素有：

* symbol: 图元的图形类别。
* symbolSize: 图元的大小。
* color: 图元的颜色。
* colorAlpha: 图元的颜色的透明度。
* opacity: 图元以及其附属物（如文字标签）的透明度。
* colorLightness: 颜色的明暗度
* colorSaturation: 颜色的饱和度
* colorHue: 颜色的色调

inRange 能定义目标系列（visualMap-continuous.seriesIndex）视觉形式，也同时定义了 visualMap-continuous 本身的视觉样式

visualMap: [

{

inRange: {

color: ['#121122', 'rgba(3,4,5,0.4)', 'red'],

symbolSize: [30, 100]

}

}

]

如果想分别定义 visualMap-continuous 本身的视觉样式和 目标系列 的视觉样式，则这样定义：

visualMap: [

{

target: {

inRange: {

color: ['#121122', 'rgba(3,4,5,0.4)', 'red'],

symbolSize: [60, 200]

}

},

controller: {

inRange: {

symbolSize: [30, 100]

}

}

}

]

target表示目标系列的视觉样式。

controller表示 visualMap-continuous 本身的视觉样式。

或者这样定义：

visualMap: [

{

inRange: {

color: ['#121122', 'rgba(3,4,5,0.4)', 'red'],

symbolSize: [60, 200]

},

controller: {

inRange: {

symbolSize: [30, 100]

}

}

}

]

controller和inRange并列，并覆盖了inRange中配置的选项

**视觉通道**

inRange 中，可以同时有任意几个的『视觉通道』定义

* 图形大小 symbolSize
* 透明度 opacity
* 颜色透明度 colorAlpha
* 颜色明暗度 colorLightness
* 颜色饱和度 colorSaturation
* 色调 colorHue

建议使用透明度（opacity），而非颜色透明度（colorAlpha）（前者能也同时控制图元中的附属物（如 label）的透明度，而后者只能控制图元本身颜色的透明度）。

视觉映射两种方式

* 线性映射
* 查表映射
* 线性映射

表示 series.data 中的每一个值会经过线性映射计算，从 [visualMap.min, visualMap.max] 映射到设定的 [visual value 1, visual value 2] 区间中的某一个视觉的值。

例如，设置[visualMap.min, visualMap.max]为[0, 100]，并且有 series.data: [50, 10, 100]。现将其映射到范围为 [0.4, 1] 的 opacity 上，用透明度表达数值大小。

那么 visualMap 组件会对 series.data 中的每一个 dataValue 做线性映射计算，得到一个 opacityValue。最终得到的 opacityValues 为 [0.7, 0.44, 1]。

visual 范围也可以反向，例如设定 opacity 范围为 [1, 0.4]

以下情况时，会设定为 线性映射：

* 当 visualMap 为 visualMap-continuous 时，或者
* 当 visualMap 为 visualMap-piecewise 且 未设置 visualMap-piecewise.categories 时。

视觉通道的值（visual value）以 Array 形式表示，例如：color: ['#333', '#777']。

* 对于颜色（color），使用数组表示例如：['#333', '#78ab23', 'blue']。意思就是以这三个点作为基准，形成一种『渐变』的色带，数据映射到这个色带上。
* 对于图形类别（symbol）：使用数据表示例如：['circle', 'rect', 'diamond']。与 visualMap.min 相等的值会映射到 'circle'，与 visualMap.max 相等的值会映射到 'diamond'。对于 中间的其他点，会根据他们和 visualMap.min 和 visualMap.max 的数值距离，映射到 'circle', 'rect', 'diamond' 中某个值上。这个不知道产生什么结果
* 透明度（opacity）、颜色透明度（colorAlpha）、颜色明暗度（colorLightness）、颜色饱和度（colorSaturation）取值范围是 [0, 1]。
* 色调（colorHue）取值范围是 [0, 360]。
* 图形类别symbol包括 'circle', 'rect', 'roundRect', 'triangle', 'diamond', 'pin', 'arrow', 'none'。还可以通过 'image://url' 设置为图片
* 查表映射

查表映射 表示 series.data 中的所有值（dataValue）是可枚举的，会根据给定的映射表查表得到映射结果。

例如，在 visualMap-piecewise 中，我们设定了 visualMap-piecewise.categories 为 ['Demon Hunter', 'Blademaster', 'Death Knight', 'Warden', 'Paladin']。我们有 series.data: ['Demon Hunter', 'Death Knight', 'Warden', 'Paladin']。然后我们可以定立查表映射规则：color: {'Warden': 'red', 'Demon Hunter': 'black'}，于是 visualMap 组件会按照表来将 dataValue 映射到 color。

如何设定为查表映射？当 visualMap 为 visualMap-piecewise 且 设置了 visualMap-piecewise.categories 时，会进行查表映射。

视觉通道的值（visual value）：一般使用 Object 或 Array 来表示，例如：

visualMap: {

type: 'piecewise',

// categories 定义了 visualMap-piecewise 组件显示出来的项。

categories: [

'Demon Hunter', 'Blademaster', 'Death Knight', 'Warden', 'Paladin'

],

inRange: {

// visual value 可以配成 Object：

color: {

'Warden': 'red',

'Demon Hunter': 'black',

'': 'green' // 空字串，表示除了'Warden'、'Demon Hunter'外，都对应到 'green'。

}

// visual value 也可以只配一个单值，表示所有都映射到一个值，如：

color: 'green',

// visual value 也可以配成数组，这个数组须和 categories 数组等长，

// 每个数组项和 categories 数组项一一对应：

color: ['red', 'black', 'green', 'yellow', 'white']

}

}

## 交互组件

数据区域缩放组件dataZoom

『概览数据整体，按需关注数据细节』是数据可视化的基本交互需求

dataZoom 组件对数轴axis进行数据窗口缩放、数据窗口平移操作。

可以通过 dataZoom.xAxisIndex 或 dataZoom.yAxisIndex 来指定 dataZoom 控制哪个或哪些数轴。

dataZoom 组件可同时存在多个，起到共同控制的作用。

dataZoom 的运行原理是通过『数据过滤』来达到『数据窗口缩放』的效果。

数据过滤模式的设置不同，效果也不同，参见：dataZoom.filterMode。

dataZoom 的数据窗口范围的设置，目前支持两种形式：

百分比形式：参见 dataZoom.start 和 dataZoom.end。

绝对数值形式：参见 dataZoom.startValue 和 dataZoom.endValue。

dataZoom 组件支持几种子组件：

* 内置型数据区域缩放组件（dataZoomInside）：内置于坐标系中。
* 滑动条型数据区域缩放组件（dataZoomSlider）：有单独的滑动条操作。
* 框选型数据区域缩放组件（dataZoomSelect）：全屏的选框进行数据区域缩放。入口和配置项均在 toolbox中。

option = {

xAxis: {

type: 'value'

},

yAxis: {

type: 'value'

},

dataZoom: [

{ // 这个dataZoom组件，默认控制x轴。

type: 'slider', // 这个 dataZoom 组件是 slider 型 dataZoom 组件

start: 10, // 左边在 10% 的位置。

end: 60 // 右边在 60% 的位置。

}

],

series: [

{

type: 'scatter',

itemStyle: {

opacity: 0.8

},

symbolSize: function (val) {

return val[2] \* 40;

},

data: [["14.616","7.241","0.896"],["3.958","5.701","0.955"],["2.768","8.971","0.669"],["9.051","9.710","0.171"],["14.046","4.182","0.536"],["12.295","1.429","0.962"],["4.417","8.167","0.113"],["0.492","4.771","0.785"],["7.632","2.605","0.645"],["14.242","5.042","0.368"]]

}

]

}

{

type: 'inside',

start: 1,

end: 35

}

鼠标滑轮缩放

symbolSize: symbolSize

# L7