异步数据加载和更新

异步加载

入门示例中的数据是在初始化后 `setOption` 中直接填入的，但是很多时候可能数据需要异步加载后再填入。Apache ECharts (incubating)™ 中实现异步数据的更新非常简单，在图表初始化后不管任何时候只要通过 jQuery 等工具异步获取数据后通过 `setOption` 填入数据和配置项就行。

```
var myChart = echarts.init(document.getElementById('main'));
$.get('data.json').done(function (data) { myChart.setOption({ title: { text: '异步数据加载示例' }, tooltip: {}, legend: { data:['销量'] }, xAxis: { data: ["衬衫","羊毛衫","雪纺衫","裤子","高跟鞋","袜子"] }, yAxis: {}, series: [{ name: '销量', type: 'bar', data: [5, 20, 36, 10, 10, 20] }] }); });
```

或者先设置完其它的样式，显示一个空的直角坐标轴，然后获取数据后填入数据。

```
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' }, tooltip: {}, legend: { data:['销量'] }, 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 }] }); });
```

如下：

ECharts 中在更新数据的时候需要通过`name`属性对应到相应的系列，上面示例中如果`name`不存在也可以根据系列的顺序正常更新，但是更多时候推荐更新数据的时候加上系列的`name`数据。

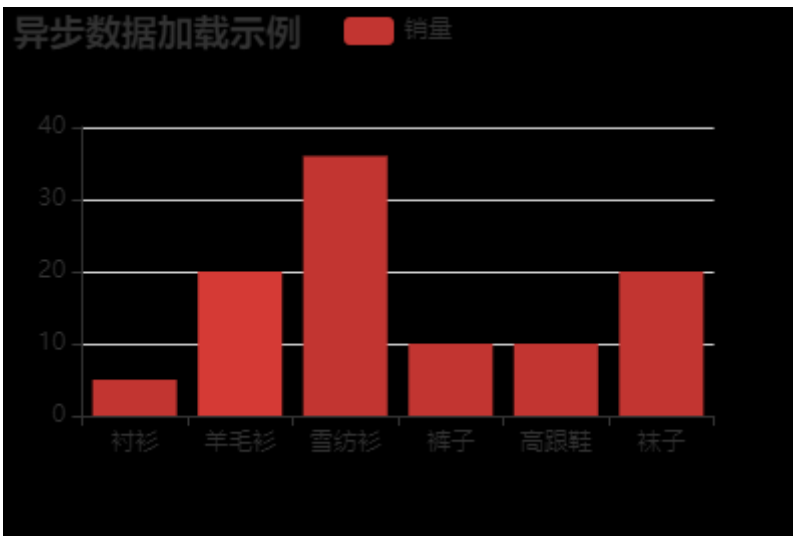
loading 动画

如果数据加载时间较长，一个空的坐标轴放在画布上也会让用户觉得是不是产生 bug 了，因此需要一个 loading 的动画来提示用户数据正在加载。

ECharts 默认有提供了一个简单的加载动画。只需要调用 `showLoading` 方法显示。数据加载完成后再调用 `hideLoading` 方法隐藏加载动画。

```
myChart.showLoading(); $.get('data.json').done(function (data) { myChart.hideLoading(); myChart.setOption(...); });
```

效果如下：



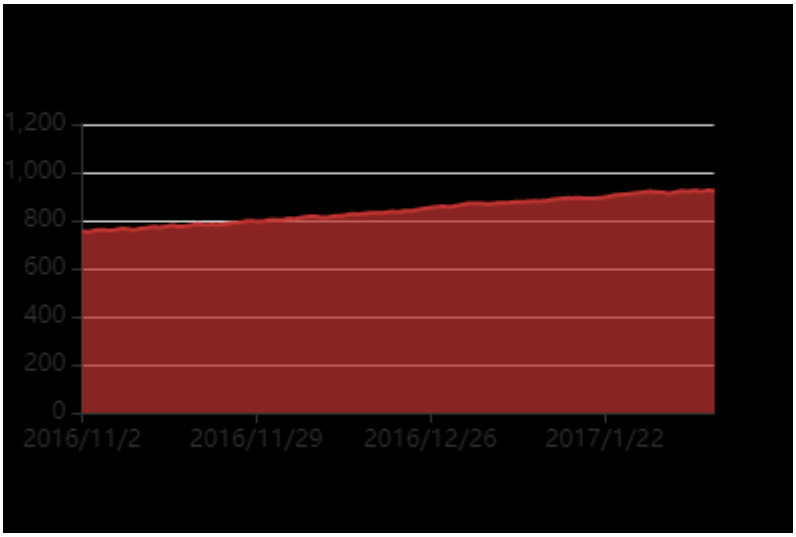
数据的动态更新

ECharts 由数据驱动，数据的改变驱动图表展现的改变，因此动态数据的实现也变得异常简单。

所有数据的更新都通过 `setOption`实现，你只需要定时获取数据，`setOption` 填入数据，而不用考虑数据到底产生了那些变化，ECharts 会找到两组数据之间的差异然后通过合适的动画去表现数据的变化。

> ECharts 3 中移除了 ECharts 2 中的 addData 方法。如果只需要加入单个数据，可以先 data.push(value) 后 setOption

具体可以看下面示例：



使用 dataset 管理数据

Apache ECharts (incubating)™ 4 开始支持了 `dataset` 组件用于单独的数据集声明，从而数据可以单独管理，被多个组件复用，并且可以基于数据指定数据到视觉的映射。这在不少场景下能带来使用上的方便。

ECharts 4 以前，数据只能声明在各个“系列（series）”中，例如：

```
option = { xAxis: { type: 'category', data: ['Matcha Latte', 'Milk Tea', 'Cheese Cocoa', 'Walnut Brownie'] }, yAxis: {}, series: [ { type: 'bar', name: '2015', data: [89.3, 92.1, 94.4, 85.4] }, { type: 'bar', name: '2016', data: [95.8, 89.4, 91.2, 76.9] }, { type: 'bar', name: '2017', data: [97.7, 83.1, 92.5, 78.1] } ] }
```

这种方式的优点是，直观易理解，以及适于对一些特殊图表类型进行一定的数据类型定制。但是缺点是，为匹配这种数据输入形式，常需要有数据处理的过程，把数据分割设置到各个系列（和类目轴）中。此外，不利于多个系列共享一份数据，也不利于基于原始数据进行图表类型、系列的映射安排。

于是，ECharts 4 提供了 数据集（`dataset`）组件来单独声明数据，它带来了这些效果：

- 能够贴近这样的数据可视化常见思维方式：(I) 提供数据，(II) 指定数据到视觉的映射，从而形成图表。
- 数据和其他配置可以被分离开来。数据常变，其他配置常不变。分开易于分别管理。
- 数据可以被多个系列或者组件复用，对于大数据量的场景，不必为每个系列创建一份数据。
- 支持更多的数据的常用格式，例如二维数组、对象数组等，一定程度上避免使用者为了数据格式而进行转换。

入门例子

下面是一个最简单的 `dataset` 的例子：

```
option = { legend: {}, tooltip: {}, dataset: { // 提供一份数据。 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] ] }, // 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。 xAxis: {type: 'category'}, // 声明一个 Y 轴，数值轴。 yAxis: {}, // 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。 series: [ {type: 'bar'}, {type: 'bar'}, {type: 'bar'} ] }
```

效果如下：

或者也可以使用常见的对象数组的格式：

```
option = { legend: {}, tooltip: {}, dataset: { // 用 dimensions 指定了维度的顺序。直角坐标系中，// 默认把第一个维度映射到 X 轴上，第二个维度映射到 Y 轴上。 // 如果不指定 dimensions，也可以通过指定 series.encode // 完成映射，参见后文。 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} ] }, xAxis: {type: 'category'}, yAxis: {}, series: [ {type: 'bar'}, {type: 'bar'}, {type: 'bar'} ] };
```

数据到图形的映射

本篇里，我们制作数据可视化图表的逻辑是这样的：基于数据，在配置项中指定如何映射到图形。

概略而言，可以进行这些映射：

- 指定 dataset 的列（column）还是行（row）映射为图形系列（series）。这件事可以使用 `series.seriesLayoutBy` 属性来配置。默认是按照列（column）来映射。
- 指定维度映射的规则：如何从 dataset 的维度（一个“维度”的意思是一行/列）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（如果有需要映射颜色大小等视觉维度的话）来配置。上面的例子中，没有给出这种映射配置，那么 ECharts 就按最常见的理解进行默认映射：X 坐标轴声明为类目轴，默认情况下会自动对应到 dataset.source 中的第一列；三个柱图系列，一一对应到 dataset.source 中后面每一列。

下面详细解释。

把数据集（dataset）的行或列映射为系列（series）

有了数据表之后，使用者可以灵活得配置：数据如何对应到轴和图形系列。

用户可以使用 `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。

看这个例子：

```
option = { legend: {}, tooltip: {}, 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] ] }, xAxis: [ {type: 'category', gridIndex: 0}, {type: 'category', gridIndex: 1} ], yAxis: [ {gridIndex: 0}, {gridIndex: 1} ], grid: [ {bottom: '55%'}, {top: '55%'} ], series: [ // 这几个系列会在第一个直角坐标系中，每个系列对应到 dataset 的每一行。 {type: 'bar', seriesLayoutBy: 'row'}, {type: 'bar', seriesLayoutBy: 'row'}, {type: 'bar', seriesLayoutBy: 'row'}, // 这几个系列会在第二个直角坐标系中，每个系列对应到 dataset 的每一列。 {type: 'bar', xAxisIndex: 1, yAxisIndex: 1}, {type: 'bar', xAxisIndex: 1,
```

```
yAxisIndex: 1}, {type: 'bar', xAxisIndex: 1, yAxisIndex: 1}, {type: 'bar', xAxisIndex: 1, yAxisIndex: 1} ] }
```

效果如下：

维度（dimension）

介绍 `encode` 之前，首先要介绍“维度（dimension）”的概念。

常用图表所描述的数据大部分是“二维表”结构，上述的例子中，我们都使用二维数组来容纳二维表。现在，当我们把系列（series）对应到“列”的时候，那么每一列就称为一个“维度（dimension）”，而每一行称为数据项（item）。反之，如果我们把系列（series）对应到表行，那么每一行就是“维度（dimension）”，每一列就是数据项（item）。

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（或者第一列）。例如上面的例子中，`'score'`、`'amount'`、`'product'` 就是维度名。从第二行开始，才是正式的数据。`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。当然也可以设置 `dataset.sourceHeader: true` 显示声明第一行（列）就是维度，或者 `dataset.sourceHeader: false` 表明第一行（列）开始就直接是数据。

维度的定义，也可以使用单独的 `dataset.dimensions` 或者 `series.dimensions` 来定义，这样可以同时指定维度名，和维度的类型（dimension type）：

```
var option1 = { dataset: { dimensions: [ {name: 'score'}, // 可以简写为 string，表示维度名。 'amount', // 可以在 type 中指定维度类型。 {name: 'product', type: 'ordinal'} ], source: [...] }, ... }; var option2 = { dataset: { source: [...] }, series: { type: 'line', // 在系列中设置的 dimensions 会更优先采纳。 dimensions: [ null, // 可以设置为 null 表示不想设置维度名 'amount', {name: 'product', type: 'ordinal'} ] }, ... };
```

大多数情况下，我们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（dimension type）可以取这些值：

- `'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。设置成 `'time'` 则能支持自动解析数据成时间戳（timestamp），比如该维度的数据是 '2017-05-10'，会自动被解析。如果这个维度被用在时间数轴（`axis.type` 为 `'time'`）上，那么会被自动设置为 `'time'` 类型。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。

数据到图形的映射（ series.encode ）

了解了维度的概念后，我们就可以使用 encode 来做映射。总体是这样的感觉：

```
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' } } ] };
```

效果如下：

`series.encode` 声明的基本结构如下，其中冒号左边是坐标系、标签等特定名称，

如 `'x'`, `'y'`, `'tooltip'` 等，冒号右边是数据中的维度名（string 格式）或者维度的序号（number 格式，从 0 开始计数），可以指定一个或多个维度（使用数组）。通常情况下，下面各种信息不需要所有的都写，按需写即可。

下面是 `series.encode` 支持的属性：

```
// 在任何坐标系和系列中，都支持： encode: { // 使用 "名为 product 的维度" 和 "名为 score 的维度" 的值
在 tooltip 中显示 tooltip: ['product', 'score'] // 使用 "维度 1" 和 "维度 3" 的维度名连起来作为系列
名。（有时候名字比较长，这可以避免在 series.name 重复输入这些名字） seriesName: [1, 3], // 表示使用
"维度2" 中的值作为 id。这在使用 setOption 动态更新数据时有用处，可以使新老数据用 id 对应起来，从而能够产生
合适的数据更新动画。 itemId: 2, // 指定数据项的名称使用 "维度3" 在饼图等图表中有用，可以使这个名字显示在图
例（legend）中。 itemName: 3 } // 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： x: [1, 5, 'score'], // 把"维度0"映射到 Y 轴。 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 }
```

下面给出个更丰富的 `series.encode` 的示例：

视觉通道（颜色、尺寸等）的映射

我们可以使用 `visualMap` 组件进行视觉通道的映射。详见 `visualMap` 文档的介绍。这是一个示例：

默认的 encode

值得一提的是，当 `series.encode` 并没有指定时，ECharts 针对最常见直角坐标系中的图表（折线图、柱状图、散点图、K线图等）、饼图、漏斗图，会采用一些默认的映射规则。默认的映射规则比较简单，大体是：

- 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）。

默认的规则不能满足要求时，就可以自己来配置 `encode`，也并不复杂。

几个常见的 series.encode 设置方式举例

问：如何把第三列设置为 X 轴，第五列设置为 Y 轴？

答：

```
series: { // 注意维度序号（dimensionIndex）从 0 开始计数，第三列是 dimensions[2]。 encode: {x: 2,
y: 4}, ... }
```

问：如何把第三行设置为 X 轴，第五行设置为 Y 轴？

答：

```
series: { encode: {x: 2, y: 4}, seriesLayoutBy: 'row', ... }
```

问：如何把第二列设置为标签？

答：关于标签的显示 `label.formatter`，现在支持引用特定维度的值，例如：

```
series: { label: { // `'{@score}'` 表示“名为 score”的维度里的值。 // `'{@[4]}'` 表示引用序号为 4 的维度里的值。 formatter: 'aaa{@product}bbb{@score}ccc{@[4]}ddd' } }
```

问：如何让第 2 列和第 3 列显示在提示框（tooltip）中？

答：

```
series: { encode: { tooltip: [1, 2] ... }, ... }
```

问：数据里没有维度名，那么怎么给出维度名？

答：

```
dataset: { dimensions: ['score', 'amount'], source: [ [89.3, 3371], [92.1, 8123], [94.4, 1954], [85.4, 829] ] }
```

问：如何把第三列映射为气泡图的点的大小？

答：

```
var option = { dataset: { source: [ [12, 323, 11.2], [23, 167, 8.3], [81, 284, 12], [91, 413, 4.1], [13, 287, 13.5] ] }, visualMap: { show: false, dimension: 2, // 指向第三列（列序号从 0 开始记，所以设置为 2）。 min: 2, // 需要给出数值范围，最小数值。 max: 15, // 需要给出数值范围，最大数值。 inRange: { // 气泡尺寸：5 像素到 60 像素。 symbolSize: [5, 60] } }, xAxis: {}, yAxis: {}, series: { type: 'scatter' } };
```

问：encode 里指定了映射，但是不管用？

答：可以查查有没有拼错，比如，维度名是：`'Life Expectancy'`，encode 中拼成了 `'Life Expectency'`。

数据的各种格式

多数常见图表中，数据适于用二维表的形式描述。广为使用的数据表格软件（如 MS Excel、Numbers）或者关系数据数据库都是二维表。他们的数据可以导出成 JSON 格式，输入到 `dataset.source` 中，在不少情况下可以免去一些数据处理的步骤。

> 假如数据导出成 csv 文件，那么可以使用一些 csv 工具如 dsv 或者 PapaParse 将 csv 转成 JSON。

在 JavaScript 常用的数据传输格式中，二维数组可以比较直观的存储二维表。前面的示例都是使用二维数组表示。

除了二维数组以外，dataset 也支持例如下面 key-value 方式的数据格式，这类格式也非常常见。但是这类格式中，目前并不支持 `seriesLayoutBy` 参数。

```
dataset: [{ // 按行的 key-value 形式（对象数组），这是个比较常见的格式。 source: [ {product: 'Matcha Latte', count: 823, score: 95.8}, {product: 'Milk Tea', count: 235, score: 81.4}, {product: 'Cheese Cocoa', count: 1042, score: 91.2}, {product: 'Walnut Brownie', count: 988, score: 76.9} ] }, { // 按列的 key-value 形式。 source: { 'product': ['Matcha Latte', 'Milk Tea', 'Cheese Cocoa', 'Walnut Brownie'], 'count': [823, 235, 1042, 988], 'score': [95.8, 81.4, 91.2, 76.9] } }]
```

多个 dataset 以及如何引用他们

可以同时定义多个 dataset。系列可以通过 `series.datasetIndex` 来指定引用哪个 dataset。例如：

```
var option = { dataset: [{ // 序号为 0 的 dataset。 source: [...], }, { // 序号为 1 的 dataset。 source: [...] }, { // 序号为 2 的 dataset。 source: [...] }], series: [{ // 使用序号
```

```
为 2 的 dataset。 datasetIndex: 2 }, { // 使用序号为 1 的 dataset。 datasetIndex: 1 }] }
```

ECharts 3 的数据设置方式（series.data）仍正常使用

ECharts 4 之前一直以来的数据声明方式仍然被正常支持，如果系列已经声明了 `series.data`， 那么就会使用 `series.data` 而非 `dataset`。

```
{ xAxis: { type: 'category' data: ['Matcha Latte', 'Milk Tea', 'Cheese Cocoa', 'Walnut Brownie'] }, yAxis: {}, series: [{ type: 'bar', name: '2015', data: [89.3, 92.1, 94.4, 85.4] }, { type: 'bar', name: '2016', data: [95.8, 89.4, 91.2, 76.9] }, { type: 'bar', name: '2017', data: [97.7, 83.1, 92.5, 78.1] }] }
```

其实，`series.data` 也是种会一直存在的重要设置方式。一些特殊的非 table 格式的图表，如 `treemap`、`graph`、`lines` 等，现在仍不支持在 dataset 中设置，仍然需要使用 `series.data`。另外，对于巨大数据量的渲染（如百万以上的数据量），需要使用 `appendData` 进行增量加载，这种情况不支持使用 `dataset`。

其他

目前并非所有图表都支持 dataset。支持 dataset 的图表有： `line`、`bar`、`pie`、`scatter`、`effectScatter`、`parallel`、`candlestick`、`map`、`funnel`、`custom`。 后续会有更多的图表进行支持。

最后，给出一个示例，多个图表共享一个 `dataset`，并带有联动交互：

ECharts 中的事件和行为

在 Apache ECharts (incubating)™ 的图表中用户的操作将会触发相应的事件。开发者可以监听这些事件，然后通过回调函数做相应的处理，比如跳转到一个地址，或者弹出对话框，或者做数据下钻等等。

在 ECharts 3 中绑定事件跟 2 一样都是通过 `on` 方法，但是事件名称比 2 更加简单了。ECharts 3 中，事件名称对应 DOM 事件名称，均为小写的字符串，如下是一个绑定点击操作的示例。

```
myChart.on('click', function (params) { // 控制台打印数据的名称 console.log(params.name); });
```

在 ECharts 中事件分为两种类型，一种是用户鼠标操作点击，或者 hover 图表的图形时触发的事件，还有一种是用户在使用可以交互的组件后触发的行为事件，例如在切换图例开关时触发的 `'legendselectchanged'` 事件（这里需要注意切换图例开关是不会触发`'legendselected'`事件的），数据区域缩放时触发的 `'datazoom'` 事件等等。

鼠标事件的处理

ECharts 支持常规的鼠标事件类型，包括 `'click'`、`'dblclick'`、`'mousedown'`、`'mousemove'`、`'mouseup'`、`'mouseover'`、`'mouseout'`、`'globalout'`、`'contextmenu'` 事件。下面先来看一个简单的点击柱状图后打开相应的百度搜索页面的示例。

```
// 基于准备好的dom，初始化ECharts实例 var myChart = echarts.init(document.getElementById('main')); // 指定图表的配置项和数据 var option = { xAxis: { data: ["衬衫","羊毛衫","雪纺衫","裤子","高跟鞋","袜子"] }, yAxis: {}, series: [{ name: '销量', type: 'bar', data: [5, 20, 36, 10, 10, 20] }] }; // 使用刚指定的配置项和数据显示图表。 myChart.setOption(option); // 处理点击事件并且跳转到相应的百度搜索页面 myChart.on('click', function (params) { window.open('https://www.baidu.com/s?wd=' + encodeURIComponent(params.name)); });
```

所有的鼠标事件包含参数 `params`，这是一个包含点击图形的数据信息的对象，如下格式：

```
{ // 当前点击的图形元素所属的组件名称， // 其值如 'series'、'markLine'、'markPoint'、'timeLine' 等。 componentType: string, // 系列类型。值可能为：'line'、'bar'、'pie' 等。当 componentType 为 'series' 时有意义。 seriesType: string, // 系列在传入的 option.series 中的 index。当 componentType 为 'series' 时有意义。 seriesIndex: number, // 系列名称。当 componentType 为 'series' 时有意义。 seriesName: string, // 数据名，类目名 name: string, // 数据在传入的 data 数组中的 index dataIndex: number, // 传入的原始数据项 data: Object, // sankey、graph 等图表同时含有 nodeData 和 edgeData 两种 data， // dataType 的值会是 'node' 或者 'edge'，表示当前点击在 node 还是 edge 上。 // 其他大部分图表中只有一种 data，dataType 无意义。 dataType: string, // 传入的数据值 value: number|Array // 数据图形的颜色。当 componentType 为 'series' 时有意义。 color: string }
```

如何区分鼠标点击到了哪里：

```
myChart.on('click', function (params) { if (params.componentType === 'markPoint') { // 点击到了 markPoint 上 if (params.seriesIndex === 5) { // 点击到了 index 为 5 的 series 的 markPoint 上。 } } else if (params.componentType === 'series') { if (params.seriesType === 'graph') { if (params.dataType === 'edge') { // 点击到了 graph 的 edge (边) 上。 } else { // 点击到了 graph 的 node (节点) 上。 } } } });
```

使用 `query` 只对指定的组件的图形元素的触发回调：

```
chart.on(eventName, query, handler);
```

`query` 可为 `string` 或者 `Object`。

如果为 `string` 表示组件类型。格式可以是 'mainType' 或者 'mainType.subType'。例如：

```
chart.on('click', 'series', function () {...}); chart.on('click', 'series.line', function () {...}); chart.on('click', 'dataZoom', function () {...}); chart.on('click', 'xAxis.category', function () {...});
```

如果为 `Object`，可以包含以下一个或多个属性，每个属性都是可选的：

```
{ <mainType>Index: number // 组件 index <mainType>Name: string // 组件 name <mainType>Id: string // 组件 id dataIndex: number // 数据项 index name: string // 数据项 name dataType: string // 数据项 type，如关系图中的 'node', 'edge' element: string // 自定义系列中的 el 的 name }
```

例如：

```
chart.setOption({ // ... series: [{ name: 'uuu' // ... }] }); chart.on('mouseover', {seriesName: 'uuu'}, function () { // series name 为 'uuu' 的系列中的图形元素被 'mouseover' 时，此方法被回调。 });
```

例如：

```
chart.setOption({ // ... series: [{ // ... }, { // ... data: [ {name: 'xx', value: 121}, {name: 'yy', value: 33} ] }] }); chart.on('mouseover', {seriesIndex: 1, name: 'xx'}, function () { // series index 1 的系列中的 name 为 'xx' 的元素被 'mouseover' 时，此方法被回调。 });
```

例如：

```
chart.setOption({ // ... series: [{ type: 'graph', nodes: [{name: 'a', value: 10}, {name: 'b', value: 20}], edges: [{source: 0, target: 1}] }] }); chart.on('click', {dataType: 'node'}, function () { // 关系图的节点被点击时此方法被回调。 }); chart.on('click', {dataType: 'edge'}, function () { // 关系图的边被点击时此方法被回调。 });
```

例如：

```
chart.setOption({ // ... series: { // ... type: 'custom', renderItem: function (params, api) { return { type: 'group', children: [{ type: 'circle', name: 'my_el', // ... }, { // ... }] } }, data: [[12, 33]] } }) chart.on('mouseup', {element: 'my_el'}, function () { // name 为 'my_el' 的元素被 'mouseup' 时，此方法被回调。 });
```

你可以在回调函数中获得这个对象中的数据名、系列名称后在自己的数据仓库中索引得到其它的信息候更新图表，显示浮层等等，如下示例代码：

```
myChart.on('click', function (parmas) { $.get('detail?q=' + params.name, function (detail) { myChart.setOption({ series: [{ name: 'pie', // 通过饼图表现单个柱子中的数据分布 data: [detail.data] }] }); }); });
```

组件交互的行为事件

在 ECharts 中基本上所有的组件交互行为都会触发相应的事件，常用的事件和事件对应参数在 `events` 文档中有列出。

下面是监听一个图例开关的示例：

```
// 图例开关的行为只会触发 legendselectchanged 事件 myChart.on('legendselectchanged', function (params) { // 获取点击图例的选中状态 var isSelected = params.selected[params.name]; // 在控制台中打印 console.log((isSelected ? '选中了' : '取消选中了') + '图例' + params.name); // 打印所有图例的状态 console.log(params.selected); });
```

代码触发 ECharts 中组件的行为

上面提到诸如`'legendselectchanged'`事件会由组件交互的行为触发，那除了用户的交互操作，有时候也会有需要在程序里调用方法触发图表的行为，诸如显示 tooltip，选中图例。

在 ECharts 2.x 是通过 `myChart.component.tooltip.showTip` 这种形式调用相应的接口触发图表行为，入口很深，而且涉及到内部组件的组织。相对地，在 ECharts 3 里改为通过调用 `myChart.dispatchAction({ type: '' })` 触发图表行为，统一管理了所有动作，也可以方便地根据需要去记录用户的行为路径。

常用的动作和动作对应参数在 `action` 文档中有列出。

下面示例演示了如何通过`dispatchAction`去轮流高亮饼图的每个扇形。