echarts

全局 echarts 对象，在 script 标签引入 `echarts.js` 文件后获得，或者在 AMD 环境中通过 `require('echarts')` 获得。

1. echarts.**init** Function

```
(dom: HTMLDivElement|HTMLCanvasElement, theme?: Object|string, opts?: {
    devicePixelRatio?: number
    renderer?: string
    width?: number|string
    height? number|string
}) => ECharts
```

创建一个 ECharts 实例，返回 echartsInstance，不能在单个容器上初始化多个 ECharts 实例。

参数

- `dom`

  实例容器，一般是一个具有高宽的 `div` 元素。

  注：如果 `div` 是隐藏的，ECharts 可能会获取不到 `div` 的高宽导致初始化失败，这时候可以明确指定 `div` 的 `style.width` 和 `style.height`，或者在 `div` 显示后手动调用 echartsInstance.resize 调整尺寸。

  ECharts 3 中支持直接使用 `canvas` 元素作为容器，这样绘制完图表可以直接将 canvas 作为图片应用到其它地方，例如在 WebGL 中作为贴图，这跟使用 echartsInstance.getDataURL 生成图片链接相比可以支持图表的实时刷新。

- `theme`

  应用的主题。可以是一个主题的配置对象，也可以是使用已经通过 echarts.registerTheme 注册的主题名称。

- `opts`

  附加参数。有下面几个可选项：

  - `devicePixelRatio`

    设备像素比，默认取浏览器的值 `window.devicePixelRatio`。

  - `renderer`

    渲染器，支持 `'canvas'` 或者 `'svg'`。参见 使用 Canvas 或者 SVG 渲染。

  - `width`

可显式指定实例宽度，单位为像素。如果传入值为 `null/undefined/'auto'`，则表示自动取 `dom`（实例容器）的宽度。

- `height`

  可显式指定实例高度，单位为像素。如果传入值为 `null/undefined/'auto'`，则表示自动取 `dom`（实例容器）的高度。

2. echarts.**connect** Function

```
(group:string|Array)
```

多个图表实例实现联动。

参数：

- `group` group 的 id，或者图表实例的数组。

示例：

```
// 分别设置每个实例的 group id
chart1.group = 'group1';
chart2.group = 'group1';
echarts.connect('group1');
// 或者可以直接传入需要联动的实例数组
echarts.connect([chart1, chart2]);
```

3. echarts.**disconnect** Function

```
(group:string)
```

解除图表实例的联动，如果只需要移除单个实例，可以将通过将该图表实例 `group` 设为空。

参数：

- `group`

  group 的 id。

4. echarts.**dispose** Function

```
(target: ECharts|HTMLDivElement|HTMLCanvasElement)
```

销毁实例，实例销毁后无法再被使用。

5. echarts.**getInstanceByDom** Function

```
(target: HTMLDivElement|HTMLCanvasElement) => ECharts
```

获取 dom 容器上的实例。

6. echarts.**registerMap** Function

```
(mapName: string, geoJson: Object, specialAreas?: Object)
```

注册可用的地图，必须在包括 geo 组件或者 map 图表类型的时候才能使用。

使用方法见 option.geo。

参数：

- `mapName`

  地图名称，在 geo 组件或者 map 图表类型中设置的 `map` 对应的就是该值。

- `geoJson`

  GeoJson 格式的数据，具体格式见 http://geojson.org/。

- `specialAreas`

  可选。将地图中的部分区域缩放到合适的位置，可以使得整个地图的显示更加好看。

  示例 USA Population Estimates：

```
echarts.registerMap('USA', usaJson, {
  // 把阿拉斯加移到美国主大陆左下方
  Alaska: {
      // 左上角经度
      left: -131,
      // 左上角纬度
      top: 25,
      // 经度横跨的范围
      width: 15
  },
  // 夏威夷
  Hawaii: {
      left: -110,
      top: 28,
      width: 5
  },
  // 波多黎各
  'Puerto Rico': {
      left: -76,
      top: 26,
      width: 2
  }
});
```

7. echarts.**getMap** Function

```
(mapName: string) => Object
```

获取已注册的地图，返回的对象类型如下

```
{
    // 地图的 geoJson 数据
    geoJson: Object,
```

```
    // 地图的特殊区域，见 registerMap
    specialAreas: Object
}
```

8. echarts.**registerTheme** Function

```
(themeName: string, theme: Object)
```

注册主题，用于初始化实例的时候指定。

9. echarts.**graphic** *

图形相关帮助方法。

echarts.graphic.**clipPointsByRect** Function

输入一组点，和一个矩形，返回被矩形截取过的点。

```
(
    // 要被截取的点列表，如 [[23, 44], [12, 15], ...]。
    points: Array.<Array.<number>>,
    // 用于截取点的矩形。
    rect: {
        x: number,
        y: number,
        width: number,
        height: number
    }
) => Array.<Array.<number>> // 截取结果。
```

echarts.graphic.**clipRectByRect** Function

输入两个矩形，返回第二个矩形截取第一个矩形的结果。

```
(
    // 要被截取的矩形。
    targetRect: {
        x: number,
        y: number,
        width: number,
        height: number
    },
    // 用于截取点的矩形。
    rect: {
        x: number,
        y: number,
        width: number,
        height: number
    }
) => { // 截取结果。
    x: number,
    y: number,
    width: number,
    height: number
}
```

echartsInstance

通过 echarts.init 创建的实例。

1. echartsInstance.**group** string, number

图表的分组，用于联动

2. echartsInstance.**setOption** Function

```
(option: Object, notMerge?: boolean, lazyUpdate?: boolean)
or
(option: Object, opts?: Object)
```

设置图表实例的配置项以及数据，万能接口，所有参数和数据的修改都可以通过 `setOption` 完成，ECharts 会合并新的参数和数据，然后刷新图表。如果开启动画的话，ECharts 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。

如下示例：

注： ECharts 2.x 中的通过 `addData` , `setSeries` 方法设置配置项的方式将不再支持，在 ECharts 3 中统一使用 `setOption`，可以参考上面示例。

参数：

调用方式：

```
chart.setOption(option, notMerge, lazyUpdate);
```

或者

```
chart.setOption(option, {
    notMerge: ...,
    lazyUpdate: ...,
    silent: ...
});
```

- `option`

  图表的配置项和数据，具体见配置项手册。

- `notMerge`

  可选，是否不跟之前设置的 `option` 进行合并，默认为 `false`，即合并。

- `lazyUpdate`

  可选，在设置完 `option` 后是否不立即更新图表，默认为 `false`，即立即更新。

- `silent`

可选，阻止调用 `setOption` 时抛出事件，默认为 `false`，即抛出事件。

3. echartsInstance.**getWidth** Function

```
() => number
```

获取 ECharts 实例容器的宽度。

4. echartsInstance.**getHeight** Function

```
() => number
```

获取 ECharts 实例容器的高度。

5. echartsInstance.**getDom** Function

```
() => HTMLCanvasElement|HTMLDivElement
```

获取 ECharts 实例容器的 dom 节点。

6. echartsInstance.**getOption** Function

```
() => Object
```

获取当前实例中维护的`option`对象，返回的`option`对象中包含了用户多次`setOption`合并得到的配置项和数据，也记录了用户交互的状态，例如图例的开关，数据区域缩放选择的范围等等。所以从这份 option 可以恢复或者得到一个新的一模一样的实例。

注意：返回的 option 每个组件的属性值都统一是一个数组，不管`setOption`传进来的时候是单个组件的对象还是多个组件的数组。如下形式：

```
{
    title: [{...}],
    legend: [{...}],
    grid: [{...}]
}
```

另外**不推荐**下面这种写法：

```
var option = myChart.getOption();
option.visualMap[0].inRange.color = ...;
myChart.setOption(option);
```

因为`getOption`获取的是已经合并过默认值了的，所以在修改了某些配置项后会导致原本是根据这些配置项值去设置的默认值失效。

因此我们更**推荐**通过`setOption`去修改部分配置。

```
myChart.setOption({
    visualMap: {
        inRange: {
            color: ...
        }
    }
})
```

7. echartsInstance.**resize** Function

```
(opts?: {
```

```
    width?: number|string,
    height?: number|string,
    silent?: boolean
}) => ECharts
```

改变图表尺寸，在容器大小发生改变时需要手动调用。

参数

- `opts`

  opts 可缺省。有下面几个可选项：

  - `width`

    可显式指定实例宽度，单位为像素。如果传入值为 `null/undefined/'auto'`，则表示自动取 `dom`（实例容器）的宽度。

  - `height`

    可显式指定实例高度，单位为像素。如果传入值为 `null/undefined/'auto'`，则表示自动取 `dom`（实例容器）的高度。

  - `silent`

    是否禁止抛出事件。默认为 `false`。

Tip: 有时候图表会放在多个标签页里，那些初始隐藏的标签在初始化图表的时候因为获取不到容器的实际高宽，可能会绘制失败，因此在切换到该标签页时需要手动调用 `resize` 方法获取正确的高宽并且刷新画布，或者在 `opts` 中显示指定图表高宽。

8. echartsInstance.**dispatchAction** Function

```
(payload: Object)
```

触发图表行为，例如图例开关 `legendToggleSelect`，数据区域缩放 `dataZoom`，显示提示框 `showTip` 等等，更多见 action 和 events 的文档。

`payload` 参数可以通过 `batch` 属性同时触发多个行为。

注：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。因此在 ECharts 3 里统一改为 `dispatchAction` 的形式。

示例

```
myChart.dispatchAction({
    type: 'dataZoom',
    start: 20,
```

```
    end: 30
});
// 可以通过 batch 参数批量分发多个 action
myChart.dispatchAction({
    type: 'dataZoom',
    batch: [{
        // 第一个 dataZoom 组件
        start: 20,
        end: 30
    }, {
        // 第二个 dataZoom 组件
        dataZoomIndex: 1,
        start: 10,
        end: 20
    }]
})
```

9. echartsInstance.**on** Function

```
(eventName: string, handler: Function, context?: Object)
```

绑定事件处理函数。

ECharts 中的事件有两种，一种是鼠标事件，在鼠标点击某个图形上会触发，还有一种是 调用 dispatchAction 后触发的事件。每个 action 都会有对应的事件，具体见 action 和 events 的文档。

如果事件是外部 dispatchAction 后触发，并且 action 中有 batch 属性触发批量的行为，则相应的响应事件参数里也会把属性都放在 batch 属性中。

参数：

- `eventName`

  事件名称，全小写，例如`'click'`，`'mousemove'`, `'legendselected'`

  注： ECharts 2.x 中会使用 `config` 对象中的 `CLICK` 等属性作为事件名称。在 ECharts 3 中统一使用跟 dom 事件一样的全小写字符串作为事件名。

- `handler`

  事件处理函数。格式为:

  ```
  (event: Object)
  ```

- `context`

  可选。回调函数内部的 `context`，即 `this` 的指向。

10. echartsInstance.**off** Function

```
(eventName: string, handler?: Function)
```

解绑事件处理函数。

参数：

- `eventName`

  事件名称。

- `handler`

  可选，可以传入需要解绑的处理函数，不传的话解绑所有该类型的事件函数。

11. echartsInstance.**convertToPixel** Function

```
(
    // finder 用于指示『使用哪个坐标系进行转换』。
    // 通常地，可以使用 index 或者 id 或者 name 来定位。
    finder: {
        seriesIndex?: number,
        seriesId?: string,
        seriesName?: string,
        geoIndex?: number,
        geoId?: string,
        geoName?: string,
        xAxisIndex?: number,
        xAxisId?: string,
        xAxisName?: string,
        yAxisIndex?: number,
        yAxisId?: string,
        yAxisName?: string,
        gridIndex?: number,
        gridId?: string
        gridName?: string
    },
    // 要被转换的值。
    value: Array|string
    // 转换的结果为像素坐标值，以 echarts 实例的 dom 节点的左上角为坐标 [0, 0] 点。
) => Array|string
```

转换坐标系上的点到像素坐标值。

例：

在地理坐标系（geo）上，把某个点的经纬度坐标转换成为像素坐标：

```
// [128.3324, 89.5344] 表示 [经度，纬度]。
// 使用第一个 geo 坐标系进行转换：
chart.convertToPixel('geo', [128.3324, 89.5344]); // 参数 'geo' 等同于 {geoIndex: 0}
// 使用第二个 geo 坐标系进行转换：
chart.convertToPixel({geoIndex: 1}, [128.3324, 89.5344]);
```

```
// 使用 id 为 'bb' 的 geo 坐标系进行转换：
chart.convertToPixel({geoId: 'bb'}, [128.3324, 89.5344]);
```

在直角坐标系（cartesian，grid）上，把某个点的坐标转换成为像素坐标：

```
// [300, 900] 表示该点 x 轴上对应刻度值 300，y 轴上对应刻度值 900。
// 注意，一个 grid 可能含有多个 xAxis 和多个 yAxis，任何一对 xAxis-yAxis 形成一个 cartesian。
// 使用第三个 xAxis 和 id 为 'y1' 的 yAxis 形成的 cartesian 进行转换：
chart.convertToPixel({xAxisIndex: 2, yAxisId: 'y1'}, [300, 900]);
// 使用 id 为 'g1' 的 grid 的第一个 cartesian 进行转换：
chart.convertToPixel({gridId: 'g1'}, [300, 900]);
```

把某个坐标轴的点转换成像素坐标：

```
// id 为 'x0' 的 xAxis 的刻度 3000 位置所对应的横向像素位置：
chart.convertToPixel({xAxisId: 'x0'}, 3000); // 返回一个 number。
// 第二个 yAxis 的刻度 600 位置所对应的纵向像素位置：
chart.convertToPixel({yAxisIndex: 1}, 600); // 返回一个 number。
```

把关系图（graph）的点转换成像素坐标：

```
// 因为每个 graph series 自己持有一个坐标系，所以我们直接在 finder 中指定 series：
chart.convertToPixel({seriesIndex: 0}, [2000, 3500]);
chart.convertToPixel({seriesId: 'k2'}, [100, 500]);
```

在某个系列所在的坐标系（无论是 cartesian、geo、graph 等）中，转换某点成像素坐标：

```
// 使用第一个系列对应的坐标系：
chart.convertToPixel({seriesIndex: 0}, [128.3324, 89.5344]);
// 使用 id 为 'k2' 的系列所对应的坐标系：
chart.convertToPixel({seriesId: 'k2'}, [128.3324, 89.5344]);
```

12. echartsInstance.**convertFromPixel** Function

```
(
    // finder 用于指示『使用哪个坐标系进行转换』。
    // 通常地，可以使用 index 或者 id 或者 name 来定位。
    finder: {
        seriesIndex?: number,
        seriesId?: string,
        seriesName?: string,
        geoIndex?: number,
        geoId?: string,
        geoName?: string,
        xAxisIndex?: number,
        xAxisId?: string,
        xAxisName?: string,
        yAxisIndex?: number,
        yAxisId?: string,
        yAxisName?: string,
        gridIndex?: number,
        gridId?: string
        gridName?: string
    },
```

```
    // 要被转换的值，为像素坐标值，以 echarts 实例的 dom 节点的左上角为坐标 [0, 0] 点。
    value: Array|string
    // 转换的结果，为逻辑坐标值。
) => Array|string
```

转换像素坐标值到逻辑坐标系上的点。是 convertToPixel 的逆运算。 具体实例可参考 convertToPixel。

13. echartsInstance.**containPixel** Function

```
(
    // finder 用于指示『在哪个坐标系或者系列上判断』。
    // 通常地，可以使用 index 或者 id 或者 name 来定位。
    finder: {
        seriesIndex?: number,
        seriesId?: string,
        seriesName?: string,
        geoIndex?: number,
        geoId?: string,
        geoName?: string,
        xAxisIndex?: number,
        xAxisId?: string,
        xAxisName?: string,
        yAxisIndex?: number,
        yAxisId?: string,
        yAxisName?: string,
        gridIndex?: number,
        gridId?: string
        gridName?: string
    },
    // 要被判断的点，为像素坐标值，以 echarts 实例的 dom 节点的左上角为坐标 [0, 0] 点。
    value: Array
) => boolean
```

判断给定的点是否在指定的坐标系或者系列上。

目前支持在这些坐标系和系列上进行判断：grid, polar, geo, series-map, series-graph, series-pie。

例：

```
// 判断 [23, 44] 点是否在 geoIndex 为 0 的 geo 坐标系上。
chart.containPixel('geo', [23, 44]); // 'geo' 等同于 {geoIndex: 0}
// 判断 [23, 44] 点是否在 gridId 为 'z' 的 grid 上。
chart.containPixel({gridId: 'z'}, [23, 44]);
// 判断 [23, 44] 点是否在 index 为 1，4，5 的系列上。
chart.containPixel({seriesIndex: [1, 4, 5]}, [23, 44]);
// 判断 [23, 44] 点是否在 index 为 1，4，5 的系列或者 gridName 为 'a' 的 grid 上。
chart.containPixel({seriesIndex: [1, 4, 5], gridName: 'a'}, [23, 44]);
```

14. echartsInstance.**showLoading** Function

```
(type?: string, opts?: Object)
```

显示加载动画效果。可以在加载数据前手动调用改接口显示加载动画，在数据加载完成后调用 hideLoading 隐藏加载动画。

参数：

- `type`

  可选，加载动画类型，目前只有一种`'default'`

- `opts`

  可选，加载动画配置项，跟 `type` 有关，下面是默认配置项：

```
default: {
  text: 'loading',
  color: '#c23531',
  textColor: '#000',
  maskColor: 'rgba(255, 255, 255, 0.8)',
  zlevel: 0
}
```

15. echartsInstance.**hideLoading** Function

隐藏动画加载效果。

16. echartsInstance.**getDataURL** Function

```
(opts: {
    // 导出的格式，可选 png, jpeg
    type?: string,
    // 导出的图片分辨率比例，默认为 1。
    pixelRatio?: number,
    // 导出的图片背景色，默认使用 option 里的 backgroundColor
    backgroundColor?: string,
    // 忽略组件的列表，例如要忽略 toolbox 就是 ['toolbox']
    excludeComponents?: Array.<string>
}) => string
```

导出图表图片，返回一个 base64 的 URL，可以设置为 `Image` 的 `src`。

示例：

```
var img = new Image();
img.src = myChart.getDataURL({
    pixelRatio: 2,
    backgroundColor: '#fff'
});
```

17. echartsInstance.**getConnectedDataURL** *

```
(opts: {
    // 导出的格式，可选 png, jpeg
    type?: string,
    // 导出的图片分辨率比例，默认为 1。
```

```
    pixelRatio?: number,
    // 导出的图片背景色，默认使用 option 里的 backgroundColor
    backgroundColor?: string,
    // 忽略组件的列表，例如要忽略 toolbox 就是 ['toolbox']
    excludeComponents?: Array.<string>
}) => string
```

导出联动的图表图片，返回一个 base64 的 url，可以设置为 `Image` 的 `src`。导出图片中每个图表的相对位置跟容器的相对位置有关。

18. echartsInstance.**appendData** *

```
(opts: {
    // 要增加数据的系列序号。
    seriesIndex?: string,
    // 增加的数据。
    data?: Array|TypedArray,
}) => string
```

此接口用于，在大数据量（百万以上）的渲染场景，分片加载数据和增量渲染。在大数据量的场景下（例如地理数的打点），就算数据使用二进制格式，也会有几十或上百兆，在互联网环境下，往往需要分片加载。`appendData` 接口提供了分片加载后增量渲染的能力，渲染新加如的数据块时不会清除原有已经渲染的部分。

注意：

- 现在不支持 系列（series） 使用 dataset 同时使用 `appendData`，只支持系列使用自己的 series.data 时使用 `appendData`。
- 目前并非所有的图表都支持分片加载时的增量渲染。目前支持的图有：ECharts 基础版本的 散点图（scatter）和 线图(lines)。ECharts GL 的 散点图(scatterGL)、线图(linesGL) 和 可视化建筑群(polygons3D)。

19. echartsInstance.**clear** *

清空当前实例，会移除实例中所有的组件和图表。清空后调用 getOption 方法返回一个`{}`空对象。

20. echartsInstance.**isDisposed** *

```
() => boolean
```

当前实例是否已经被释放。

21. echartsInstance.**dispose** *

销毁实例，销毁后实例无法再被使用。

action

ECharts 中支持的图表行为，通过 dispatchAction 触发。

注： 代码中的 `?:` 表示该属性是可选的。*EVENT:* 是 action 对应触发的事件。

1. action.**highlight** Action

高亮指定的数据图形。

通过 `seriesName` 或者 `seriesIndex` 指定系列。如果要再指定某个数据可以再指定 `dataIndex` 或者 `name`。

```
dispatchAction({
    type: 'highlight',
    // 可选，系列 index，可以是一个数组指定多个系列
    seriesIndex?: number|Array,
    // 可选，系列名称，可以是一个数组指定多个系列
    seriesName?: string|Array,
    // 可选，数据的 index
    dataIndex?: number,
    // 可选，数据的 名称
    name?: string
})
```

2. action.**downplay** Action

取消高亮指定的数据图形。

通过 `seriesName` 或者 `seriesIndex` 指定系列。如果要指定某个数据可以再指定 `dataIndex` 或者 `name`。

```
dispatchAction({
    type: 'downplay',
    // 可选，系列 index，可以是一个数组指定多个系列
    seriesIndex?: number|Array,
    // 可选，系列名称，可以是一个数组指定多个系列
    seriesName?: string|Array,
    // 可选，数据的 index
    dataIndex?: number,
    // 可选，数据的 名称
    name?: string
})
```

3. action.**legend** *

图例组件相关的行为，必须引入图例组件后才能使用。

action.legend.**legendSelect** Action

选中图例。

```
dispatchAction({
    type: 'legendSelect',
    // 图例名称
    name: string
})
```

EVENT: legendselected
action.legend.**legendUnSelect** Action

取消选中图例。

```
dispatchAction({
    type: 'legendUnSelect',
```

```
    // 图例名称
    name: string
})
```

EVENT: legendunselected
action.legend.**legendToggleSelect** Action

切换图例的选中状态。

```
dispatchAction({
    type: 'legendToggleSelect',
    // 图例名称
    name: string
})
```

EVENT: legendselectchanged
action.legend.**legendScroll** Action

控制图例的滚动。当 legend.type 为 `'scroll'` 时有效。

```
dispatchAction({
    type: 'legendScroll',
    scrollDataIndex: number,
    legendId: string
})
```

EVENT: legendscroll
4. action.**tooltip** *
提示框组件相关的行为，必须引入提示框组件后才能使用。
action.tooltip.**showTip** Action

显示提示框。

有下面两种使用方式。

1 指定在相对容器的位置处显示提示框，如果指定的位置无法显示则无效。

```
dispatchAction({
    type: 'showTip',
    // 屏幕上的 x 坐标
    x: number,
    // 屏幕上的 y 坐标
    y: number,
    // 本次显示 tooltip 的位置。只在本次 action 中生效。
    // 缺省则使用 option 中定义的 tooltip 位置。
    position: Array.<number>|string|Function
})
```

2 指定数据图形，根据 tooltip 的配置项显示提示框。

```
dispatchAction({
```

```
    type: 'showTip',
    // 系列的 index，在 tooltip 的 trigger 为 axis 的时候可选。
    seriesIndex?: number,
    // 数据的 index，如果不指定也可以通过 name 属性根据名称指定数据
    dataIndex?: number,
    // 可选，数据名称，在有 dataIndex 的时候忽略
    name?: string,
    // 本次显示 tooltip 的位置。只在本次 action 中生效。
    // 缺省则使用 option 中定义的 tooltip 位置。
    position: Array.<number>|string|Function,
})
```

参数 `position` 同 tooltip.position 相同。
action.tooltip.**hideTip** Action

隐藏提示框。

```
dispatchAction({
    type: 'hideTip'
})
```

5. action.**dataZoom** *
数据区域缩放组件相关的行为，必须引入数据区域缩放组件后才能使用。
action.dataZoom.**dataZoom** Action

数据区域缩放。

```
dispatchAction({
    type: 'dataZoom',
    // 可选，dataZoom 组件的 index，多个 dataZoom 组件时有用，默认为 0
    dataZoomIndex: number,
    // 开始位置的百分比，0 - 100
    start: number,
    // 结束位置的百分比，0 - 100
    end: number,
    // 开始位置的数值
    startValue: number,
    // 结束位置的数值
    endValue: number
})
```

EVENT: datazoom

6. action.**visualMap** *

视觉映射组件相关的行为，必须引入视觉映射组件后才能使用。
action.visualMap.**selectDataRange** Action

选取映射的数值范围。

```
dispatchAction({
```

```
    type: 'selectDataRange',
    // 可选，visualMap 组件的 index，多个 visualMap 组件时有用，默认为 0
    visualMapIndex: number,
    // 连续型 visualMap 和 离散型 visualMap 不一样
    // 连续型的是一个表示数值范围的数组。
    // 离散型的是一个对象，键值是类目或者分段的索引。值是 `true`, `false`
    selected: Object|Array
})
```

å 示例：

```
myChart.dispatchAction({
    type: 'selectDataRange',
    // 选取 20 到 40 的值范围
    selected: [20, 40],
    // 取消选中第二段
    selected: { 1: false },
    // 取消选中类目 `优`
    selected: { '优': false }
});
```

EVENT: datarangeselected

7. action.**timeline** *

时间轴组件相关的行为，必须引入时间轴组件后才能使用。

action.timeline.**timelineChange** Action

设置当前的时间点。

```
dispatchAction({
    type: 'timelineChange',
    // 时间点的 index
    currentIndex: number
})
```

EVENT: timelinechanged

action.timeline.**timelinePlayChange** Action

切换时间轴的播放状态。

```
dispatchAction({
    type: 'timelinePlayChange',
    // 播放状态，true 为自动播放
    playState: boolean
})
```

EVENT: timelineplaychanged

8. action.**toolbox** *

工具栏组件相关的行为，必须引入工具栏组件后才能使用。

action.toolbox.**restore** Action

重置 option。

```
dispatchAction({
    type: 'restore'
})
```

EVENT: restore

9. action.**pie** *

饼图相关的行为，必须引入饼图后才能使用。

action.pie.**pieSelect** Action

选中指定的饼图扇形。

```
dispatchAction({
    type: 'pieSelect',
    // 可选，系列 index，可以是一个数组指定多个系列
    seriesIndex?: number|Array,
    // 可选，系列名称，可以是一个数组指定多个系列
    seriesName?: string|Array,
    // 数据的 index，如果不指定也可以通过 name 属性根据名称指定数据
    dataIndex?: number,
    // 可选，数据名称，在有 dataIndex 的时候忽略
    name?: string
})
```

EVENT: pieselected

action.pie.**pieUnSelect** Action

取消选中指定的饼图扇形。

```
dispatchAction({
    type: 'pieUnSelect',
    // 可选，系列 index，可以是一个数组指定多个系列
    seriesIndex?: number|Array,
    // 可选，系列名称，可以是一个数组指定多个系列
    seriesName?: string|Array,
    // 数据的 index，如果不指定也可以通过 name 属性根据名称指定数据
    dataIndex?: number,
    // 可选，数据名称，在有 dataIndex 的时候忽略
    name?: string
})
```

EVENT: pieunselected

action.pie.**pieToggleSelect** Action

切换指定的饼图扇形选中状态。

```
dispatchAction({
    type: 'pieToggleSelect',
```

```
    // 可选，系列 index，可以是一个数组指定多个系列
    seriesIndex?: number|Array,
    // 可选，系列名称，可以是一个数组指定多个系列
    seriesName?: string|Array,
    // 数据的 index，如果不指定也可以通过 name 属性根据名称指定数据
    dataIndex?: number,
    // 可选，数据名称，在有 dataIndex 的时候忽略
    name?: string
})
```

EVENT: pieselectchanged

10. action.**geo** *

地图组件相关的行为，必须引入地图组件后才能使用。

action.geo.**geoSelect** Action

选中指定的地图区域。

```
dispatchAction({
    type: 'geoSelect',
    // 可选，系列 index，可以是一个数组指定多个系列
    seriesIndex?: number|Array,
    // 可选，系列名称，可以是一个数组指定多个系列
    seriesName?: string|Array,
    // 数据的 index，如果不指定也可以通过 name 属性根据名称指定数据
    dataIndex?: number,
    // 可选，数据名称，在有 dataIndex 的时候忽略
    name?: string
})
```

EVENT: geoselected

action.geo.**geoUnSelect** Action

取消选中指定的地图区域。

```
dispatchAction({
    type: 'geoUnSelect',
    // 可选，系列 index，可以是一个数组指定多个系列
    seriesIndex?: number|Array,
    // 可选，系列名称，可以是一个数组指定多个系列
    seriesName?: string|Array,
    // 数据的 index，如果不指定也可以通过 name 属性根据名称指定数据
    dataIndex?: number,
    // 可选，数据名称，在有 dataIndex 的时候忽略
    name?: string
})
```

EVENT: geounselected

action.geo.**geoToggleSelect** Action

切换指定的地图区域选中状态。

```
dispatchAction({
    type: 'geoToggleSelect',
    // 可选，系列 index，可以是一个数组指定多个系列
    seriesIndex?: number|Array,
    // 可选，系列名称，可以是一个数组指定多个系列
    seriesName?: string|Array,
    // 数据的 index，如果不指定也可以通过 name 属性根据名称指定数据
    dataIndex?: number,
    // 可选，数据名称，在有 dataIndex 的时候忽略
    name?: string
})
```

EVENT: geoselectchanged
11. action.**map** *

地图图表相关的行为，必须引入地图图表后才能使用。

。
action.map.**mapSelect** Action

选中指定的地图区域。

```
dispatchAction({
    type: 'mapSelect',
    // 可选，系列 index，可以是一个数组指定多个系列
    seriesIndex?: number|Array,
    // 可选，系列名称，可以是一个数组指定多个系列
    seriesName?: string|Array,
    // 数据的 index，如果不指定也可以通过 name 属性根据名称指定数据
    dataIndex?: number,
    // 可选，数据名称，在有 dataIndex 的时候忽略
    name?: string
})
```

EVENT: mapselected
action.map.**mapUnSelect** Action

取消选中指定的地图区域。

```
dispatchAction({
    type: 'mapUnSelect',
    // 可选，系列 index，可以是一个数组指定多个系列
    seriesIndex?: number|Array,
    // 可选，系列名称，可以是一个数组指定多个系列
    seriesName?: string|Array,
    // 数据的 index，如果不指定也可以通过 name 属性根据名称指定数据
    dataIndex?: number,
    // 可选，数据名称，在有 dataIndex 的时候忽略
    name?: string
```

```
})
```

EVENT: mapunselected
action.map.**mapToggleSelect** Action

切换指定的地图区域选中状态。

```
dispatchAction({
    type: 'mapToggleSelect',
    // 可选，系列 index，可以是一个数组指定多个系列
    seriesIndex?: number|Array,
    // 可选，系列名称，可以是一个数组指定多个系列
    seriesName?: string|Array,
    // 数据的 index，如果不指定也可以通过 name 属性根据名称指定数据
    dataIndex?: number,
    // 可选，数据名称，在有 dataIndex 的时候忽略
    name?: string
})
```

EVENT: mapselectchanged
12. action.**graph** *

关系图 相关的行为，必须引入 关系图 后才能使用。

。
action.graph.**focusNodeAdjacency** Action

将指定的节点以及其所有邻接节点高亮。

```
dispatchAction({
    type: 'focusNodeAdjacency',

    // 使用 seriesId 或 seriesIndex 或 seriesName 来定位 series.
    seriesId: 'xxx',
    seriesIndex: 0,
    seriesName: 'nnn',

    // 使用 dataIndex 来定位节点。
    dataIndex: 12
})
```

最后会抛出 focusNodeAdjacency 事件。
action.graph.**unfocusNodeAdjacency** Action

将指定的节点以及其所有邻接节点高亮。

```
dispatchAction({
    type: 'unfocusNodeAdjacency',

    // 使用 seriesId 或 seriesIndex 或 seriesName 来定位 series.
    seriesId: 'xxx',
```

```
    seriesIndex: 0,
    seriesName: 'nnn'
})
```

最后会抛出 unfocusNodeAdjacency 事件。

13. action.**brush** *

区域选择相关的行为。

action.brush.**brush** *

触发此 action 可向 echarts 中添加一个或多个选框，例如：

```
myChart.dispatchAction({
    type: 'brush',
    areas: [ // areas 表示选框的集合，可以指定多个选框。
        { // 选框一：
            geoIndex: 0, // 指定此选框属于 index 为 0 的 geo 坐标系。
                         // 也可以通过 xAxisIndex 或 yAxisIndex 来指定此选框属于直角坐标系。
                         // 如果没有指定，则此选框属于『全局选框』。不属于任何坐标系。
                         // 属于『坐标系选框』，可以随坐标系一起缩放平移。属于全局的选框不行。
            brushType: 'polygon', // 指定选框的类型。还可以为 'rect', 'lineX', 'lineY'
            range: [ // 如果是『全局选框』，则使用 range 来描述选框的范围。
                ...
            ],
            coordRange: [ // 如果是『坐标系选框』，则使用 coordRange 来指定选框的范围。
                [119.72,34.85],[119.68,34.85],[119.5,34.84],[119.19,34.77]
                // 这个例子中，因为指定了 geoIndex，所以 coordRange 里单位是经纬度。
            ]
        },
        ... // 选框二、三、四、...
    ]
});
```

其中，`areas` 中的 `range` 和 `coordRange` 的格式，根据 brushType 不同而不同：

- brushType 为 'rect' `range` 和 `coordRange` 的格式为：`[[minX, maxX], [minY, maxY]]`
- brushType 为 'lineX' 或 'lineY' `range` 和 `coordRange` 的格式为：[min, maxX]
- brushType 为 'polygon' `range` 和 `coordRange` 的格式为：[[point1X, point1X], [point2X, point2X], ...]

`range` 和 `coordRange` 的区别是：

- 当此选框为『全局选框』时，使用 `range`。
- 当此选框为『坐标系选框』时（即指定了 `geoIndex` 或 `xAxisIndex` 或 `yAxisIndex` 时），使用 `coordRange`。
- `range` 的单位为 *像素*，`coordRange` 的单位为 *坐标系单位*，比如 geo 中，`coordRange` 单位为经纬度，直角坐标系中，coordRange 单位为对应轴的数据的单位。

events

在 ECharts 中主要通过 on 方法添加事件处理函数，该文档描述了所有 ECharts 的事件列表。

ECharts 中的事件分为两种，一种是鼠标事件，在鼠标点击某个图形上会触发，还有一种是 调用 dispatchAction 后触发的事件。

示例：

```
myChart.on('click', function (params) {
    console.log(params);
});

myChart.on('legendselectchanged', function (params) {
    console.log(params);
});
```

1. events.**鼠标事件** *

鼠标事件的事件参数是事件对象的数据的各个属性，对于图表的点击事件，基本参数如下，其它图表诸如饼图可能会有部分附加参数。例如饼图会有 `percent` 属性表示百分比，具体见各个图表类型的 label formatter 回调函数的 `params`。

```
{
    // 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
}
```

鼠标事件包括`'click'`，`'dblclick'`，`'mousedown'`，`'mouseup'`，`'mouseover'`，`'mouseout'`，`'globalout'`，`'contextmenu'`。

参见 ECharts 中的事件和行为

2. events.**legendselectchanged** Event

ACTION: legendToggleSelect 切换图例选中状态后的事件。

注：图例组件用户切换图例开关会触发该事件。

```
{
    type: 'legendselectchanged',
    // 切换的图例名称
    name: string
    // 所有图例的选中状态表
    selected: Object
}
```

3. events.**legendselected** Event

ACTION: legendSelect 图例选中后的事件。

```
{
    type: 'legendselected',
    // 选中的图例名称
    name: string
    // 所有图例的选中状态表
    selected: Object
}
```

注： ECharts 2.x 中用户开关图例对应的事件从 `legendselected` 改为 legendselectchanged。

4. events.**legendunselected** Event

ACTION: legendUnSelect 图例取消选中后的事件。

```
{
    type: 'legendunselected',
    // 取消选中的图例名称
    name: string
    // 所有图例的选中状态表。
    selected: Object
}
```

5. events.**legendscroll** Event

ACTION: legendscroll 图例滚动事件。

```
{
    type: 'legendscroll',
    scrollDataIndex: number
    legendId: string
}
```

6. events.**datazoom** Event

ACTION: dataZoom

数据区域缩放后的事件。

```
{
    type: 'datazoom',
    // 缩放的开始位置的百分比，0 - 100
    start: number
    // 缩放的结束位置的百分比，0 - 100
    end: number
    // 缩放的开始位置的数值，只有在工具栏的缩放行为的事件中存在。
    startValue?: number
    // 缩放的结束位置的数值，只有在工具栏的缩放行为的事件中存在。
    endValue?: number
}
```

7. events.**datarangeselected** Event

ACTION: selectDataRange 视觉映射组件中，`range` 值改变后触发的事件。

```
{
    type: 'datarangeselected',
    // 连续型 visualMap 和 离散型 visualMap 不一样
    // 连续型的是一个表示数值范围的数组。
    // 离散型的是一个对象，键值是类目或者分段的索引。值是`true`或`false`
    selected: Object|Array
}
```

8. events.**timelinechanged** Event

ACTION: timelineChange 时间轴中的时间点改变后的事件。

```
{
    type: 'timelinechanged',
    // 时间点的 index
    currentIndex: number
}
```

9. events.**timelineplaychanged** Event

ACTION: timelinePlayChange 时间轴中播放状态的切换事件。

```
{
    type: 'timelineplaychanged',
    // 播放状态，true 为自动播放
    playState: boolean
}
```

10. events.**restore** Event

ACTION: restore 重置 option 事件。

```
{
    type: 'restore'
}
```

11. events.**dataviewchanged** Event

工具栏中数据视图的修改事件。

```
{
```

```
    type: 'dataviewchanged'
}
```

12. events.**magictypechanged** Event

工具栏中动态类型切换的切换事件。

```
{
    type: 'magictypechanged',
    // 点击切换的当前类型，同 echarts 2.x 中的 type 属性
    currentType: string
}
```

13. events.**geoselectchanged** Event

ACTION: geoToggleSelect

geo 中地图区域切换选中状态的事件。

用户点击选中会触发该事件。

```
{
    type: 'geoselectchanged',
    // 系列 ID，可以在 option 中传入
    seriesId: string
    // 数据名称
    name: name,
    // 所有数据的选中状态表。
    selected: Object
}
```

14. events.**geoselected** Event

ACTION: geoSelect

geo 中地图区域选中后的事件。

使用 `dispatchAction` 可触发此事件，用户点击不会触发此事件（用户点击事件请使用 geoselectchanged）。

```
{
    type: 'geoselected',
    // 系列 ID，可以在 option 中传入
    seriesId: string
    // 数据名称
    name: name,
    // 所有数据的选中状态表。
    selected: Object
}
```

15. events.**geounselected** Event

ACTION: geoUnSelect

geo 中地图区域取消选中后的事件。

使用 `dispatchAction` 可触发此事件，用户点击不会触发此事件（用户点击事件请使用 geoselectchanged）。

```
{
    type: 'geounselected',
    // 系列 ID，可以在 option 中传入
    seriesId: string
    // 数据名称
    name: name,
    // 所有数据的选中状态表。
    selected: Object
}
```

16. events.**pieselectchanged** Event

ACTION: pieToggleSelect

series-pie 中饼图扇形切换选中状态的事件。

用户点击选中会触发该事件。

```
{
    type: 'pieselectchanged',
    // 系列 ID，可以在 option 中传入
    seriesId: string
    // 数据名称
    name: name,
    // 所有数据的选中状态表。
    selected: Object
}
```

注： 该事件同 ECharts 2 中的 `pieSelected` 事件相同。

17. events.**pieselected** Event

ACTION: pieSelect

series-pie 中饼图扇形选中后的事件。

使用 `dispatchAction` 可触发此事件，用户点击不会触发此事件（用户点击事件请使用 pieselectchanged）。

```
{
    type: 'pieselected',
    // 系列 ID，可以在 option 中传入
    seriesId: string
    // 数据名称
    name: name,
    // 所有数据的选中状态表。
    selected: Object
}
```

注： ECharts 2.x 中用户开关图例对应的事件从 `pieselected` 改为 pieselectchanged。

18. events.**pieunselected** Event

ACTION: pieUnSelect

series-pie 中饼图扇形取消选中后的事件。

使用 `dispatchAction` 可触发此事件，用户点击不会触发此事件（用户点击事件请使用 pieselectchanged）。

```
{
    type: 'pieunselected',
    // 系列 ID，可以在 option 中传入
    seriesId: string
    // 数据名称
    name: name,
    // 所有数据的选中状态表。
    selected: Object
}
```

19. events.**mapselectchanged** Event

ACTION: mapToggleSelect

series-map 中地图区域切换选中状态的事件。

用户点击选中会触发该事件。

```
{
    type: 'mapselectchanged',
    // 系列 ID，可以在 option 中传入
    seriesId: string
    // 数据名称
    name: name,
    // 所有数据的选中状态表。
    selected: Object
}
```

注： 该事件同 ECharts 2 中的 `mapSelected` 事件相同。

20. events.**mapselected** Event

ACTION: mapSelect

series-map 中地图区域选中后的事件。

使用 `dispatchAction` 可触发此事件，用户点击不会触发此事件（用户点击事件请使用 mapselectchanged）。

```
{
    type: 'mapselected',
    // 系列 ID，可以在 option 中传入
    seriesId: string
    // 数据名称
    name: name,
    // 所有数据的选中状态表。
    selected: Object
```

```
}
```

注： ECharts 2.x 中用户开关图例对应的事件从 `mapselected` 改为 mapselectchanged。

21. events.**mapunselected** Event

ACTION: mapUnSelect

series-map 中地图区域取消选中后的事件。

使用 `dispatchAction` 可触发此事件，用户点击不会触发此事件（用户点击事件请使用 mapselectchanged）。

```
{
    type: 'mapunselected',
    // 系列 ID，可以在 option 中传入
    seriesId: string
    // 数据名称
    name: name,
    // 所有数据的选中状态表。
    selected: Object
}
```

22. events.**axisareaselected** Event

平行坐标轴 (Parallel)范围选取事件。

当进行坐标轴范围选取时，可以用如下方式获取当前高亮的线所对应的 data indices （即 `series` 的 `data` 中的序号列表）。

```
chart.on('axisareaselected', function () {
    var series0 = chart.getModel().getSeries()[0];
    var series1 = chart.getModel().getSeries()[1];
    var indices0 = series0.getRawIndicesByActiveState('active');
    var indices1 = series1.getRawIndicesByActiveState('active');
    console.log(indices0, indices1);
});
```

23. events.**focusnodeadjacency** Event

graph 的邻接节点高亮事件。

参见 focusNodeAdjacency。

24. events.**unfocusnodeadjacency** Event

graph 的邻接节点取消高亮事件。

参见 unfocusNodeAdjacency。

25. events.**brush** Event

选框添加事件。即发出 brush action 得到的事件。

26. events.**brushselected** Event

对外通知当前选中了什么。

参见 区域选择。

这个事件，在 `setOption` 时不会发出，在其他的 dispatchAction 时，或者用户在界面中创建、删除、修改选框时会发出。

事件参数内容为：

```
{
    type: 'brushselected',
    batch: [
        {
            brushIndex: number // brush 组件的 id，大多数情况只使用一个 brush 组件，所以不必理会。
            selected: [ // 每个系列被选中的项。
                        // 注意，如果某个系列不支持 brush，但是还是会在这里出现对应的项。
                        // 也就是说，selected 可以使用 seriesIndex 来直接找到对应的项。
                { // series 0 被选中的项
                    seriesIndex: number,
                    dataIndex: [ 3, 6, 12, 23 ] // 用这些 dataIndex，可以去原始数据中找到真正的值。
                },
                { // series 1 被选中的项
                    seriesIndex: number,
                    dataIndex: []
                },
                ...
            ]
        },
        ...
    ]
}
```

事件使用方式例如：

```
var dataBySeries = [
    [ 12, 23, 54, 6 ], // series 0 的数据
    [ 34, 34433, 2223, 21122, 1232, 34 ] // series 1 的数据
];

chart.setOption({
    ...,
    brush: {
        ...
    },
    series: [
        { // series 0
            data: dataBySeries[0]
        },
        { // series 1
            data: dataBySeries[1]
```

```
        }
    ]
});

chart.on('brushSelected', function (params) {
    var brushComponent = params.batch[0];

    var sum = 0; // 统计选中项的数据值的和

    for (var sIdx = 0; sIdx < brushComponent.selected.length; sIdx++) {
        // 对于每个 series：
        var dataIndices = brushComponent.selected[sIdx].dataIndex;

        for (var i = 0; i < dataIndices.length; i++) {
            var dataIndex = dataIndices[i];
            sum += dataBySeries[sIdx][dataIndex];
        }
    }
    console.log(sum); // 用某种方式输出统计值。
});
```

如果想**避免此事件频繁触发**，可以使用 brush.throttleType。