# Echarts基本使用

## 定义Echarts和Option

下面提供的两种方法都需要定义Echarts对象和Option。

echartsEcharts!: ECharts;

  echartsEChartsOption = {

    xAxis: {

      type: 'category',

      data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

    },

    yAxis: {

      type: 'value'

    },

    series: [

      {

        data: [120, 200, 150, 80, 70, 110, 130],

        type: 'bar',

        showBackground: true,

        backgroundStyle: {

          color: 'rgba(180, 180, 180, 0.2)'

        }

      }    ]  };

## 1.使用id获取DOM元素

使用简单，但不建议使用

Html文件代码

<div id="id"></div>

Ts文件，通过函数获取DOM，初始化图表设置数据

  getHtml() {

    let test: any = document.getElementById('id');

    this.echarts = init(test);

    this.echarts.setOption(this.EChartsOption);

  }

## 2.使用ViewChild获取DOM元素

HTML文件

<div #echrts id="echrtsView"></div>

获取DOM元素

@ViewChild('echrts') echrts: any;

初始化设置图表数据

  ngAfterViewInit(): void {

    this.echartsEcharts = init(this.echrts.nativeElement);

    this.echartsEcharts.setOption(this.echartsEChartsOption);

  }

# 父组件传值给子组件

## 创建子组件后，父组件获取数据

ngOnInit(): void {

    this.getData()

}

res: any

getData() {

    this.http.get('assets/data-directive.json').subscribe((response: any) => {

      this.res = response.data

    })

 }

## 通过input传给子组件

其中父组件html这么写：

<app-child [res]="res"></app-child>

子组件ts接收：

@Input() res: any;

## 子组件通过ViewChild获取元素绘制图表

1. 引入并使用ViewChild

Ts文件使用

@ViewChild("echart") echartdom: any;

Html标记

<div #echart class="echart">

</div>

1. 子组件引入并使用Echarts

import { ECharts, init } from "echarts";

echart!: ECharts;

1. 建立option

  EChartsOption = {

    tooltip: {

      trigger: "item",

      transitionDuration: 0,

    },

    legend: {

      orient: "vertical",

      left: "left",

    },

    series: [

      {

        name: "内存情况",

        data: [

          { value: 0 },

        ],

        type: "pie",

        radius: ["60%", "70%"],

        avoidLabelOverlap: false,

        label: {

          show: false,

          position: "center",

        },

        emphasis: {

          // label: {

          //   show: true,

          //   fontSize: '40',

          //   fontWeight: 'bold'

          // }

          itemStyle: {

            shadowBlur: 10,

            shadowOffsetX: 0,

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

          },

        },

        labelLine: {

          show: false,

        },

        itemStyle: {

          normal: {

            color: function (params) {

              var colorList = ["#F52222", "#55A722", "#1FE6D4"];

              return colorList[params.dataIndex];

            },

            label: {

              show: true,

              position: "top",

              // formatter: "{b}\n{c}",

            },

          },

        },

      },

    ],

  };

1. 在子组件初始化echarts

  ngAfterViewInit(): void {

    this.echart = init(this.echartdom.nativeElement);

  }

初始化的过程在视图即将出现的生命周期函数ngAfterViewInit中

1. 数据驱动更新

下面这是更新echarts图表的函数，在ngOnChanges生命周期函数中调用。当有数据改变时就会触发更新。

  freshEcharts() {

    let arr = [];

    arr = [

      {

        value: this.res.usedStorage,

        name: "已用内存",

      },

      {

        value: this.res.availableStorage,

        name: "可用内存",

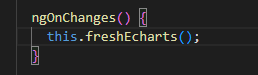
      },

    ];

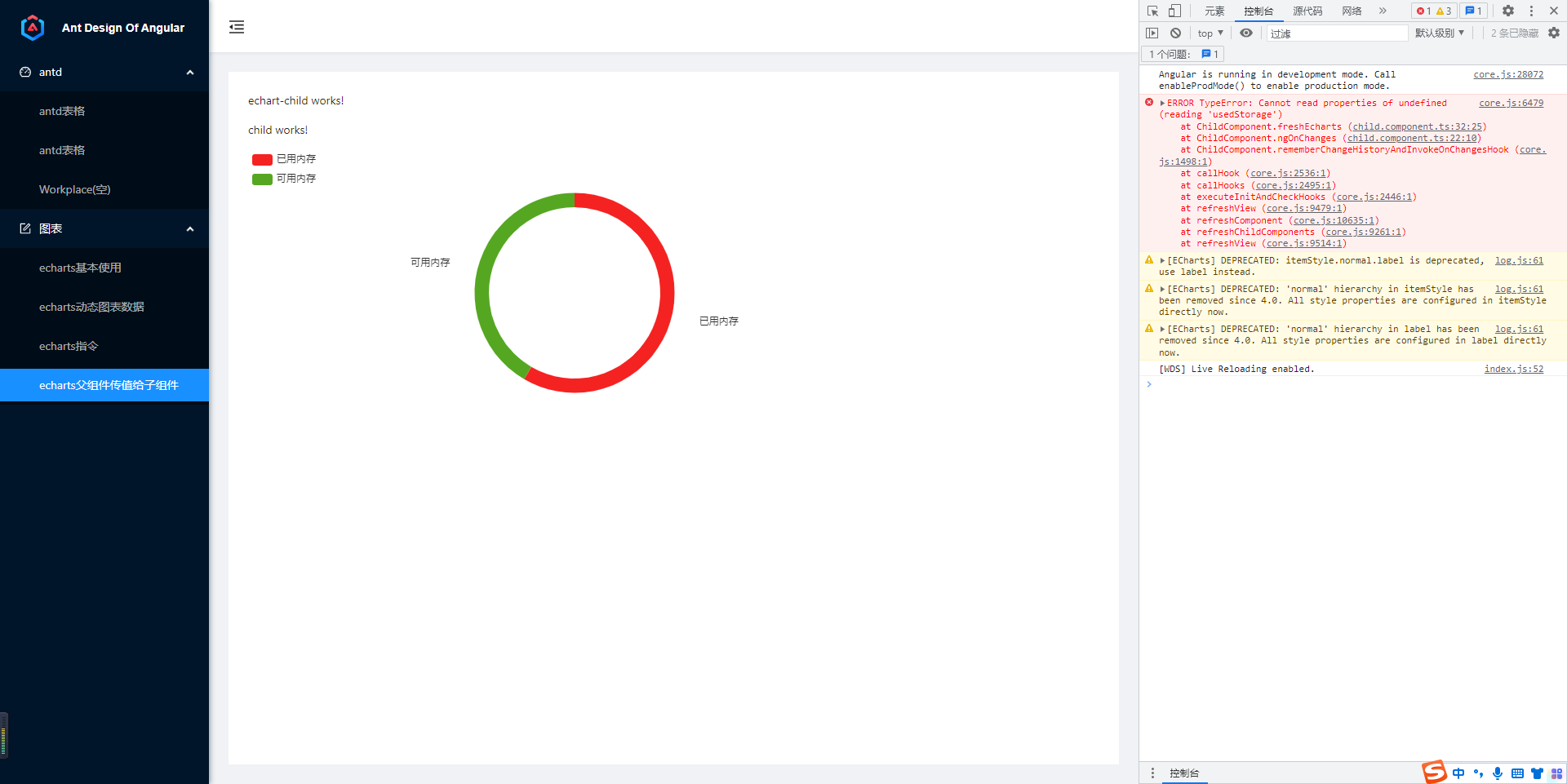
    this.EChartsOption.series[0].data = arr;

    this.echart.setOption(this.EChartsOption);

  }



1. 此时页面显示出图表，但是页面有报错



## 解决报错

（1）属性不存在

core.js:6479 ERROR TypeError: Cannot read properties of undefined (reading 'usedStorage')

我们前面是把整个返回的数据传给子组件的，这里说读取不到这个属性，那么我们可以在父组件进行定义一个空值。代码如下：

  res: any ={

    usedStorage:0,

  }

（2）无法设置图表配置项setOption

core.js:6479 ERROR TypeError: Cannot read properties of undefined (reading 'setOption')

这里就是子组件ngOnChanges使用问题了。我们打印看看这里使用的具体信息。

  ngOnChanges(changes: SimpleChanges) {

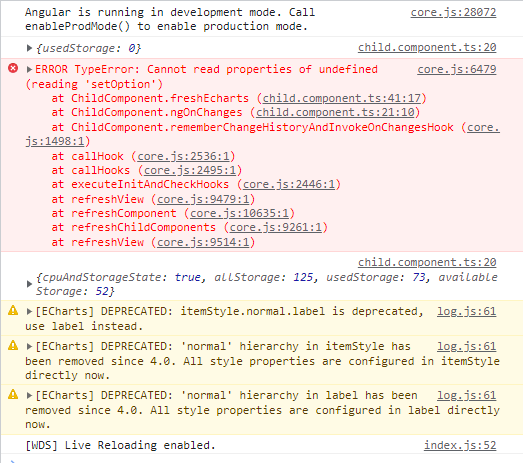
    const current = changes.res && changes.res.currentValue

    console.log(current);

    this.freshEcharts();

  }

这里打印changes表现不是特别直观，我们就打印current的值。



由打印结果可以得知，第一次打印输出为未get获取到数据的父组件初始值，第二次打印输出为正常获取到的数据，但是在第一次未获取正常数据之前，已经进行了一次更新图表，这就是报错的原因。解决方案很简单，让更新图表的函数选择性执行就行。编码思路：

我们刚刚打印得知，前后相差几个变量，那么我们判断这相差的变量是否存在即可。

代码如下：

  ngOnChanges(changes: SimpleChanges) {

    const current = changes.res && changes.res.currentValue

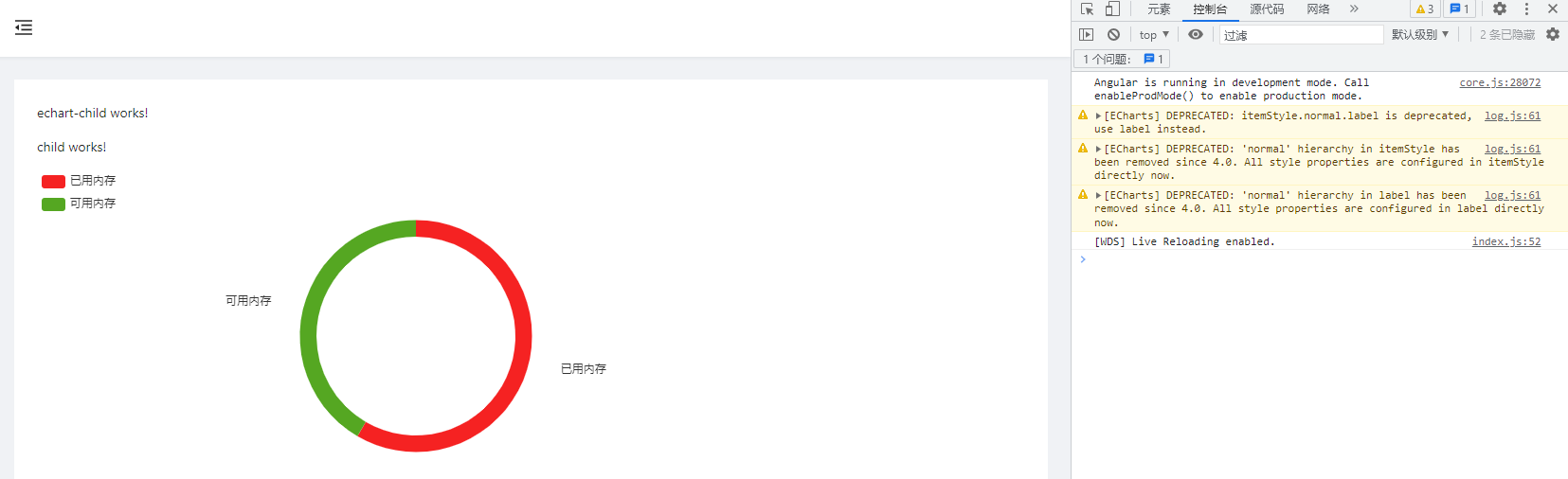
    if ('availableStorage' in current) {

      this.freshEcharts();

    }

  }

这样就可以正常显示而且没有报错了。



# Echarts使用水球图

## 下载插件

yarn add echarts-liquidfill

## 在main.ts中注册

import 'echarts-liquidfill';

## 进行使用

使用方式和echarts没有太大区别，主要是在option中修改参数。

Series数组再加入一个对象：

{

        name: "内存占用",

        outline: {

          show: false,

        },

        radius: "60%",

        type: "liquidFill",

        data: [

          {

              value: 0.6,

              direction: 'left', //波浪方向

          },

        ],

        amplitude: '15 %', //波浪的振幅

        color: ["#dbc7b5"],

        label: {

          formatter: function (param) {

            return `${param.seriesName}: ${param.value \* 100}%`;

          },

          fontSize: 14,

        },

        tooltip: {

          formatter(param) {

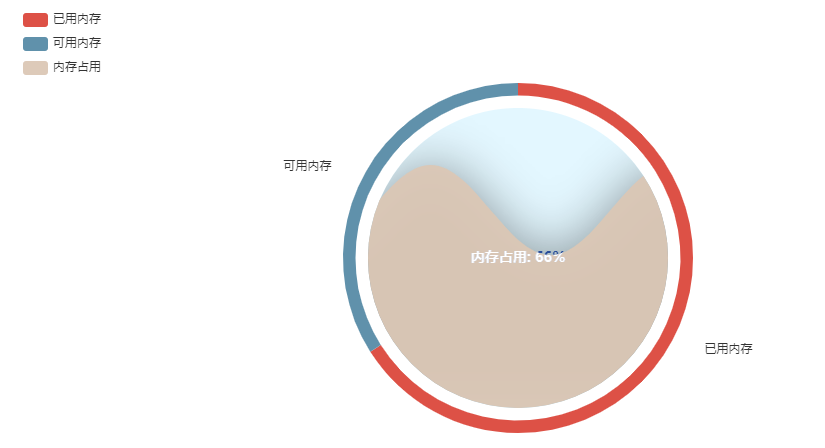
            return `${param.seriesName}: ${param.value \* 100}%`;

          },

        },

},

## 效果图如下



## 参数讲解

### 类型type: 'liquidFill',

这是水球图实现的关键，type声明了这个的图的属性。

官方讲解中这里的type有这些属性：

line（折线图）、bar（柱状图）、pie（饼图）、scatter（散点图）、effectScatter（气泡图）、radar（雷达图）、tree（树图）、treemap（树状数据图）、sunburst（旭日图）、boxplot（箱形图）、candlestick（K线图）、heatmap（热力图）、map（地图）、parallel（平行坐标系）、lines（路径图）、graph（关系图）、sankey（桑基图）、funnel（漏斗图）、gauge（仪表盘）、pictorialBar（象形柱图）、themeRiver（主题河流）、custom（自定义系列）。

### 多重波浪

通过设置data的多个值可以展现多个波浪。

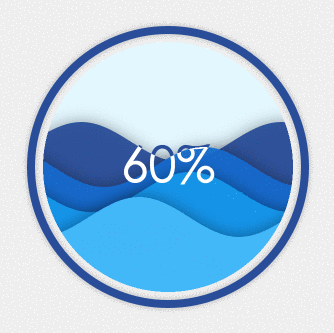
var option = {

series: [{

type: 'liquidFill',

data: [0.6, 0.5, 0.4, 0.3]

}]};



### 静态波

水波纹一般都是动态的，通过设置这三个值可以让波纹不再波动。

waveAnimation: false,

animationDuration: 0,

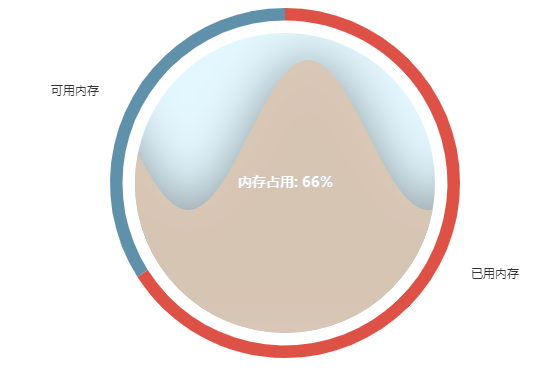
animationDurationUpdate: 0,

### 振幅amplitude

波浪的曲折程度可以理解为水波图的振幅，当振幅为0时水波纹就是平的。



设置值大的时候展现就和山峰一样。



### 背景样式backgroundStyle

backgroundStyle: {

borderWidth: 5,

borderColor: 'red',

color: 'yellow'

}

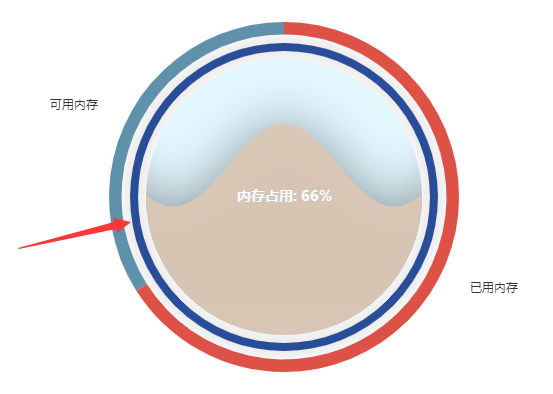
### 内环圈设置outline

outline: {

show: false

}

这里的show值显示TRUE时会显示这里箭头处的内环圈。为FALSE时则会隐藏。



### **形状Shape**

这个形状可以根据需求进行修改，比如方形、圆形、三角形等。

shape: 'diamond'



### 动画animationDurationUpdate、animationDuration

一般来说，液体填充图表中有两种类型的动画。

第一种是初始动画，有波浪上升的效果。此动画的缓动方法由控制动画制作和它的持续时间动画持续时间.

第二种是更新动画，通常在数据变化、波高变化时使用。它们由控制动画更新和动画持续更新.

参数较多，感兴趣说的伙伴可以阅读github讲解。

<https://github.com/ecomfe/echarts-liquidfill>

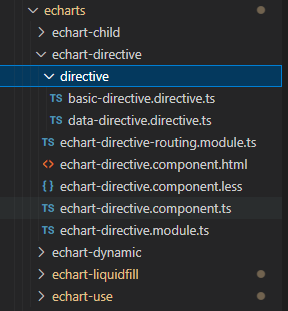
# Echarts指令

## 在目录下创建指令

ng g directive dataDirective

## 在module引入指令

创建指令时可能在module文件中进行引入，如果没有引入就需要手动引入。



我们在echart-directive文件下单独建立directive文件夹，里面创建指令文件。如果没有自动引入，这里就需要在echart-directive.module.ts文件中手动进行引入。

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { EchartDirectiveRoutingModule } from './echart-directive-routing.module';

import { EchartDirectiveComponent } from './echart-directive.component';

import { DataDirectiveDirective } from './directive/data-directive.directive';

import { BasicDirectiveDirective } from './directive/basic-directive.directive';

@NgModule({

  declarations: [

    EchartDirectiveComponent,

    DataDirectiveDirective,

    BasicDirectiveDirective,

  ],

  imports: [

    CommonModule,

    EchartDirectiveRoutingModule,

  ]

})

export class EchartDirectiveModule { }

## Html文件中使用指令

引入比较简单，在创建指令文件的时候，指令文件有这么几行代码，里面写了这个指令的名称appBasicDirective。

@Directive({

  selector: '[appBasicDirective]'

})

我们之间在div中进行引入就行。

<div appBasicDirective class="echart"></div>

## 在指令ts文件中进行编写代码

### 引入依赖

import { Directive, ElementRef } from '@angular/core';

import { EChartsOption, ECharts, init } from 'echarts';

import { fromEvent, Subject } from 'rxjs';

import { debounceTime, takeUntil } from 'rxjs/operators';

### 定义echarts对象和其他内容

  chart: ECharts;

  destroy$ = new Subject();

  simulatedData = [6, 7, 8, 9, 8]

### 构造器注册

  constructor(

    public el: ElementRef

  ) { }

### 定义Option

private \_assembleOptions(): EChartsOption {

    return {

      title: {text: '指令的基本使用' },

      xAxis: {

        type: 'category',

        data: ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

      },

      yAxis: { type: 'value' },

      series: [

        {

          data: [150, 230, 224, 218, 135, 147, 260],

          type: 'line'

        }

      ]

    };

  }

### 初始化并设置数据

  ngAfterViewInit(): void {

    this.chart = init(this.el.nativeElement);

    this.chart.setOption(this.\_assembleOptions());

    fromEvent(window, 'resize')

      .pipe(

        takeUntil(this.destroy$),

        debounceTime(2000)

      )

      .subscribe(() => {

        this.chart.resize();

      });

  }

TakeUntil 函数官方文档对这个操作符的解释是：

Emit values until provided observable emits.

即它可以被赋予另一个起锚定作用的 Observable，当该锚定 Observable emit 值时，原始的 Observable 就停止发射值，进入 complete 操作。或者说到达一个条件后取消订阅。

debounceTime函数用于防抖。

resize 函数的作用是改变图表尺寸，在容器大小发生改变时需要手动调用。

1. 销毁图表实例

 ngOnDestroy(): void {

    this.destroy$.next();

    this.destroy$.complete();

    this.chart.dispose();

  }

dispose函数的作用是销毁实例，实例销毁后无法再被使用。