<u> 下划线 </u>

一级标题

二级标题

~~删除线~~
斜体

加粗

斜体加粗

- 11
- ss
- faf

1. 哈
2. 哈哈
3. 安抚哈佛哈

```
// 代码段
let name = {};
```

[link 跳转]

CSS

@import 简化页面

- @import url() all
- @import url() screen
- @import utl() print

@media 定义局部响应式布局

- @media screen add(max-width:600){

  .nav{

  }

  }

AND 条件判断响应式

```
//多个media属性合在一起使用
```

根选择器

```
//选中html元素
:root{
    background:orange;
}
```

兄弟选择器

```
//h1后面的所有h2兄弟元素被选中
h1~h2{

}或
//h1后面的紧挨着的h2兄弟元素被选中（只会选中最近的h2）
h1+h2{

}
```

属性选择器

```
//选中有title属性的元素
h1[title]{

}
//选中title=‘wxx’的元素
h1[title='wxx']{

}

//通配符选中title含有‘w’的元素
h1[title*='w']{

}
//通配符选中title以‘w’开始的元素
h1[title~='w']{

}
```

结构伪类选择器

```
h1:last-child{}
h1:first-child{}
h1:nth-child{}
h1:nth-last-child{}
h1:nth-of-type{}
//匹配的元素的父元素中仅有一个子元素，而且是一个唯一的子元素。
h1:only-child{}
//用来选择一个元素是它的父元素的唯一一个相同类型的子元素
h1:only-of-type{}
//排除选择器（:not） ：  在前三个元素中排除掉第一个元素，选中剩余两个元素
h1:nth-child(-n+3):not(:first-child){}
```

排除选择器

```
:not(:first-child) //排除第一个元素
```

空选择器

```
//:empty选择器表示的就是空。
//用来选择没有任何内容的元素，这里没有内容指的是一点内容都没有，哪怕是一个空格。
h2:empty{
    display:none;
    border:1px solid green;
}
```

::selection 选择器

```
//“::selection”伪元素是用来匹配突出显示的文本(用鼠标选择文本时的文本)。
//浏览器默认情况下，用鼠标选择网页文本是以“深蓝的背景，白色的字体”显示的
::selection{
   background: orange;
   color: green;
}
```

样式权重

```
可以继承（继承的样式没有权重）：
    font
    color
    .
    .
不能继承：
    border
    .
    .
```

```
通配符的权重（权重为0）：
    *{

    }
通配符权重>继承的样式权重
```

```
强制权重优先级：
    .c{
        color:red !important;
    }
```

字体

```
*{
    font-style: italic
    font-size
    font-family
    line-height
    font-weight
    font-variant:lowercase/upcase/... //小型大写/首字母大写/...
}
<em>  //强调加粗
```

文本线条

```
*{
    text-decoration:none//下划线
 }
```

文本阴影

```
h2{
    text-shadow:2px 2px 2px 2px rgba(2,2,2,.2) inset //x y 模糊距离 模糊大小 颜色 向内阴影
}
```

文本溢出和空白处理技巧

```
h2{
    white-space//处理空白的样式
    white-space:nowarp//规定文本不换行
}
//注意：弹性盒场景下会失效
文本溢出省略号：
h2{
    overflow:hidden;
    text-overflow:ellipsis;
    white-space:nowrap;
}
文本与图片对齐：
img{
    vertical-align:middle/top/bottom...
}
```

文本缩进

```
h2{
    font-size:14px;
    text-indent:2em;//缩进2字符
}
h2{
    text-align:center//文本中心对齐
}
```

字符间距与排序

```
h2{
    letter-spacing:20px;//控制字符间距
    word-spacing:30px;//控制每个单词间距
    writing-mode:vertical-rl//文字从右到左排序
}
```

宽高比

```
// 设置宽高比
// 宽高比 = 1
h3{
    aspect-ratio: 1;
}
```

轮廓线

```
//轮廓线在边线外围
h2{
    outline-style //线形状
    outline-width
    outline-color
    outline:none
}
```

表格 table

**table表格属性：

属性	值	描述
Width	Pixels、%	规定表格的宽度
align	Left、center、right	表格相对周围元素的对齐方式。
border	pixels	规定表格边框的宽度。
Bgcolor	rgb(x,x,x)、#xxxxxx、Colorname	表格的背景颜色。
Cellpadding	Pixels、%	单元边沿与其内容之间的空白。
cellspacing	Pixels、%	单元格之间的空白。
frame	属性值	规定外侧边框的哪个部分是可见的。
rules	属性值	规定内侧边框的哪个部分是可见的。

```
// th的高度自适应(设置th的vertical-align)
```

```
<table>
    <tr>
        <th>电压等级</th>
        <td>作业单位作业单位作业单位作业单位作业单位作业单位作业单位</td>
        <th>关联线路</th>
        <td>作业单位作业单位作业单位作业单位作业单位</td>
    </tr>
</table>

css:
    table {
      border-collapse: collapse;
      border: 1px solid #444;
      font-size: 12px;
    }
    th,
    td {
      padding: 10px 5px;
      border: 1px solid #444;
      border-spacing: 0px;
    }
    th {
      width: 65px;
      height: 100%;
      vertical-align: middle;// 通过设置th的vertical-align,让th的文字垂直居中，自适应高度的变化
    }
```

元素显隐

```
h2{
    display:none;
    visibility:hidden;
    opacity:0;
}
```

fill-available

```
//fill-available只针对块级盒子
h2{
    height:fill-available//自动撑满
    width:max-content//宽度根据内容大小自适
}
```

xx-content

```
//fit-content自动撑开div宽度，而不至于让div独占一行
div{
    width:fit-content;
    margin:auto;
}
//max-content
div{
    width:max-content;
```

```
}

//min-content
div{
    width:min-content;
}
```

背景

```
//设置元素背景图片的原始起始位置。
background-origin:  border-box | padding-box | content-box | no-clip
```

需要注意的是，如果背景不是no-repeat，这个属性无效，它会从边框开始显示。

```
//用来将背景图片做适当的裁剪以适应实际需要。
background-clip :  border-box | padding-box | content-box | no-clip
```

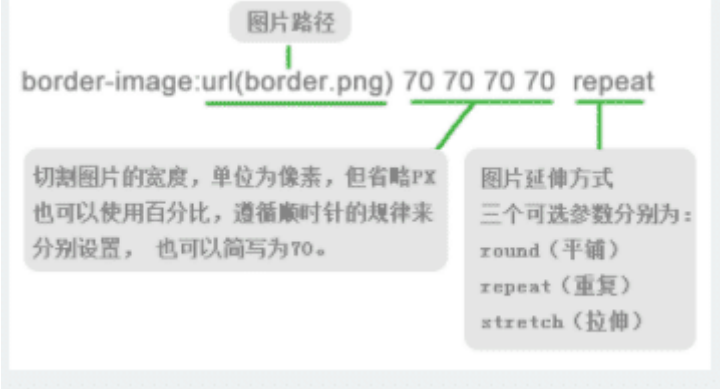

```
//背景裁切 ：背景在盒子中的显示范围
h2{
    background-clip:content-box/padding-box/border-box  //显示在(内容/padding/边框)位置
}
//背景滚动
h2{
    background-attachment:fixed/scroll  //背景在盒子中固定或者随滚动条滚动
}
//位置
h2{
    background-position: top left; //位置名字组合定位，只写一个默认另一个为居中
}
//背景重复
```

```
h2{
    background-repeat:no-repeat;//不重复
    background-size:cover/contain; //背景尺寸设置像素大小，只写一个默认另一个为auto，cover(常用)填充整个盒子，contain将图片缩放至某一边紧贴容器边缘为止
    box-shadow:0 0 5px rgb(2,2,2) ;
}
//背景颜色线性渐变 ，角度为顺时针计算
h2{
    background:linear-gradient(red,green,blue) //默认，从下向上
    background:linear-gradient(to left,red,green,blue)//线性渐变,从左向右
    background:linear-gradient(90deg,red,green,blue)//设置线性渐变顺时针旋转角度，从左向右
}
//径向渐变
h2{
    background:radial-gradient(20px 50px,red,green,blue)//径向渐变
}
//可同时设置多个背景图片
组合写法：颜色 路径 重复 位置/大小
```

边框阴影

```
h2{
    ////注意：inset 可以写在参数的第一个或最后一个，其它位置是无效的。
    //如果添加多个阴影，只需用逗号隔开即可
    //box-shadow: X轴偏移量 Y轴偏移量 [阴影模糊半径] [阴影扩展] [阴影颜色] [投影方式];
    box-shadow:0px 0px 6px 6px #ccc inset,0px 0px 6px #333;
}
```

边框背景图片




```
h2{
    border-image:url(xxxx) 20 repeat;//
}
```

清除浮动

```
//伪元素清除浮动
h2{

}
h2::after{
    content:'';
    clear:both;
    display:block;
}
//overflow清除浮动
h2{
   overflow:hidden;
}
```

定位

```
//注意：滚动条会对定位造成影响

 h2{
     position:relative/absolute/fixed
 }

//粘性定位
h2{
    position:sticky;
}
```

过渡

```
h2{
    transition-duration:2s;//设置过渡时间
    transition-property:all/none/color/width... //设置可以过渡的属性
    transition-timing-function:ease/linear/ease-in/steps(3,end)... //设置过渡的运动轨迹,steps()设置步进帧动画
    transition-delay:2s;//设置过渡动画的延迟时间
}
```

动画

```
//定义动画
@keyframes hd{
    //初始
    from{
        color:green;
    }
    50%{
        color:blue;
        transform:translateX(300px)
    }
    //终点
    to{
        color:red;
        transform:translate(200px,100px)
```

```
    }
}
h2{
    animation-name:hd;
    animation-duration:2s ,1s;//定义多个动画持续时间
    animation-delay:1s,2s;//定义多个动画的延迟时间
    animation-iteration-count:2/, infinite无限; //设置动画的执行次数，infinite无限
    animation-direction:normal/reverse/alternate;//normal和reverse设置动画执行的起始方向,alternate和alternate-reverse控制动画平滑轮回执行
    animation-timing-function:steps(4,end);//动画步进帧执行
    animation-play-state:running/pause;//设置动画的暂停、运动
    animation-fill-mode:forwards//动画填充模式
}
```

弹性布局

```
//弹性盒子为块级元素
//声明
//注意:需要弄清弹性盒(也叫弹性容器)和弹性元素的概念区别
h2{
    display:flex/inline-block;//块级弹性盒/行内弹性盒
}

//具体的弹性盒（弹性容器）属性设置
h2{
    display:flex;
    flex-direction:row/column/row-reverse/column-reverse; //设置弹性元素方向行、列、反向行、反向列
    flex-wrap:wrap/no-wrap/wrap-reverse;//弹性盒溢出换行处理
    flex-flow:row wrap  //flex-direction和flex-wrap简写

    justify-content:flex-start/center/flex-end/space-around/space-between/space-evenly //设置主轴的排列方式 space-evenly设置完全平均分布
    align-items:flex-start/flex-end/center/stretch... //设置交叉轴排列方式(flex-wrap不换行时)
    align-content:start/center/end/space-around/space-between... //设置多行元素在交叉轴的排列方式
}

//具体的弹性元素(弹性容器的子元素)属性设置
h3{
    align-self:flex-start/flex-end/stretch...;//对单个弹性元素交叉轴排列方式进行控制
    flex-grow:1/2...;//设置子元素分配剩余空间   元素会放大
    flex-grow:0;//设置该子元素不分配剩余空间
    flex-shrink:0;//设置元素缩小比例   元素会缩小
    flex-basic:100px;//设置主轴的基准尺寸为100px

    //弹性元素属性的组合定义
    flex:1 2 100px

    order：0/1/2... //控制弹性元素在弹性容器中的排序

    //注意：弹性元素设置绝对定位时，因为绝对定位后失去了正常的空间位，将不受弹性盒属性影响
}
```

滤镜

```
h2{
    filter:blur(5px);
}
```

扭曲 skew()

```
//扭曲skew()函数能够让元素倾斜显示
//这与rotate()函数的旋转不同，rotate()函数只是旋转，而不会改变元素的形状。
//skew()函数不会旋转，而只会改变元素的形状。
h2{
    transform:skew(20deg,30deg);
}
```

attr()函数

```
<h2 wxx="hahaha"></h2>

h2::after{
    content:attr(wxx);//等效与 content:'hahaha'
    ...
}
```

var()函数

```
//定义一个名为 "--main-bg-color" 的属性，然后使用 var() 函数调用该属性
:root {
  --main-bg-color: coral;
}

h2 {
  background-color: var(--main-bg-color);
}
```

小于 12px 文字生成器

```
https://qishaoxuan.github.io/css_tricks/smallFont/
//使用 svg 作为解决小于 12px 字号文字的方案：
    //1.使用 transform: scale() 设置后占位区域并没有改变，难以调节对齐方式。
    //2.使用 canvas 无法选中文字（也可以解决，但不如 svg 简洁）
```

dom 监听事件

```

```

link 详情

H5 新增标签

```
// 新增标签
```

javascript

数据类型

```
//五种基本类型
//String Number boolean undefined null

//其他都为Object

//typeof(xx)  检查数据类型

//undefined == null 结果为true

//将类型转为字符类型 toString()不能用在null、undefined上
xx.toString()、 Strign(xx)、 xx + ''

//转为是数字
Number(xx)、xx * 1、parseInt(xx)、parseFloat(xx)

//转为布尔
Boolean(xx)

//内置对象
Date Math Array Error Reg
```

数组挖掘

- arr1.copyWithin([1,2,3]) 数组复制
- find() 数组查找
- findIndex() 查找下标
- includes() 数组查找
- indexOf() 数组查找
- lastIndexOf() 从后查找数组索引
- sort(()=>a-b) 按倒叙排序
- forEach() 遍历数组，return 跳出本次循环，trycatch 结束当前循环
- every() some() 返回 boolean
- filter() 按条件过滤并返回新数组
- map() 遍历数组并返回新数组
- reduce((pre,next,index)=>{},[]) ==pre-前一次执行 reduce 返回的对象，next-本次(将要)执行 reduce 函数的对象，index 当前索引==
- pop() 删除数组最后一个元素，并返回原数组
- push() 在数组后面添加元素
- shift() 删除数组第一个元素，并返回原数组
- unshift() 在数组前面添加元素
- slice(start,opt_end) 返回新数组，并且不会改变原数组,opt_end 可以为负值
- splice(index,opt_length，opt_new) 返回新数组，并且修改原数组，opt_new 作为第三个参数时可以向原数组添加元素
-
-

清空数组

```
let hd = [1, 2];
//1.
while (hd.pop()) {}
//2.
hd = [];
//3
hd.length = 0;
```

数组拆分合并

```
let arr = [1, 2];
let hd = [3, 4];

arr = arr.concat(hd); //合并
arr = [...arr, ...hd]; //合并
arr.copyWithin(hd); //合并
```

跳出指定 for 循环

```
//为循环语句创建一个label，来标识当前的循环
outer: for (var i = 0; i < 10; i++) {
  for (var j = 0; j < 10; j++) {
    if (j == 5) {
      break outer;
    }
    console.log(j);
  }
}
```

for in 和 for of

```
// for of只能遍历可迭代的变量，所以对对象不可以使用for of
for (let key in arr) {
  //key-键
}
for (let key in obj) {
  //key-键
}
// 不可遍历对象obj
for (let value of arr) {
  //value-值
}
```

Object.entries()

```
//Object.entries()方法返回一个给定对象自身可枚举属性的键值对数组，其排列与使用for…in循环遍历该对象时返回的顺序一致。
//区别在于forin 会枚举原型链上的属性。
let per = {
    name:'zdx',
    age: 18
  }

  for(let [key,value] of Object.entries(per)){
    console.log(key,value);
  }
  //name,zdx
  //age,18
```

slice、substring、substr

```
string.slice(start, end); //提取一个字符串,end支持负数
string.substring(start, end); //提取一个字符串,end不支持负数
string.substr(start, len); //提取一个长度为len的字符串
```

forEach

```
//forEach可以通过return跳出本次循环，不可以用break,会报错
//将每次循环当作在一个匿名函数中执行
arr.forEach(item=>{
    if(false){
        return
    }
    ...
})

//可以使用trycatch结束循环
//结束整个循环
try{
    arr.forEach(item=>{
        if(**){
           throw('打印：循环结束')
           }
    })
}catch(e){
    console.log(e) //打印：循环结束
}
```

Symbol()

```
//在程序中永远不会重复
//定义Symbol
let hd = Symbol("weixiaoxiang");
console.log(hd.toString()); //weixiaoxiang
console.log(Symbol.for(hd)); //weixiaoxiang
```

Set 和 Map

```
//Set 和 Map 主要的应用场景在于 数据重组 和 数据储存
//Set 是一种叫做集合的数据结构，Map 是一种叫做字典的数据结构

//Set 类似与数组
操作方法：
    add(value)：新增，相当于 array里的push
    delete(value)：存在即删除集合中value
    has(value)：判断集合中是否存在 value
    clear()：清空集合
遍历方法：
    keys()：返回一个包含集合中所有键的迭代器
    values()：返回一个包含集合中所有值得迭代器
    entries()：返回一个包含Set对象中所有元素得键值对迭代器
    forEach(callbackFn, thisArg)：用于对集合成员执行callbackFn操作


//Map 本质是键值对的集合，类似与对象集合，通过get 和 set读取数据
操作方法：
    set(key, value)：向字典中添加新元素
    get(key)：通过键查找特定的数值并返回
    has(key)：判断字典中是否存在键key
    delete(key)：通过键 key 从字典中移除对应的数据
    clear()：将这个字典中的所有元素删除
遍历方法：
    Keys()：将字典中包含的所有键名以迭代器形式返回
    values()：将字典中包含的所有数值以迭代器形式返回
    entries()：返回所有成员的迭代器
    forEach()：遍历字典的所有成员
```

async 和 defer

```
async; //允许异步script文件
defer; //也允许异步加载script文件
```

对象

- Object.getOwnPropertyDescriptor(object1, 'property1') 获得对象属性

  ```
  {
  configurable:true,//是否可以删除`
  enumerable:true,//是否可以枚举`
  value:'',//值`
  writeable:true//是否可以修改`
  }
  ```

- delete obj.name 删除对象 name 属性
- Object.hasOwnProperty() 判断对象是否包含特定的自身（非继承）属性
-

探索 js 中函数的秘密

- call() 传参数 可以改变函数中 this 的指向 立即执行

- apply() 传数组 可以改变函数中 this 的指向 立即执行
- bind() 传参数 不立即执行
- call() bind() apply() 可以用于构造函数的继承

同步异步

- process.nexTick() setImmediate
- 同步任务>process.nexTick()>微任务>宏任务>setImmmediate() 当前事件任务循环
- async 函数返回值是 promise 对象

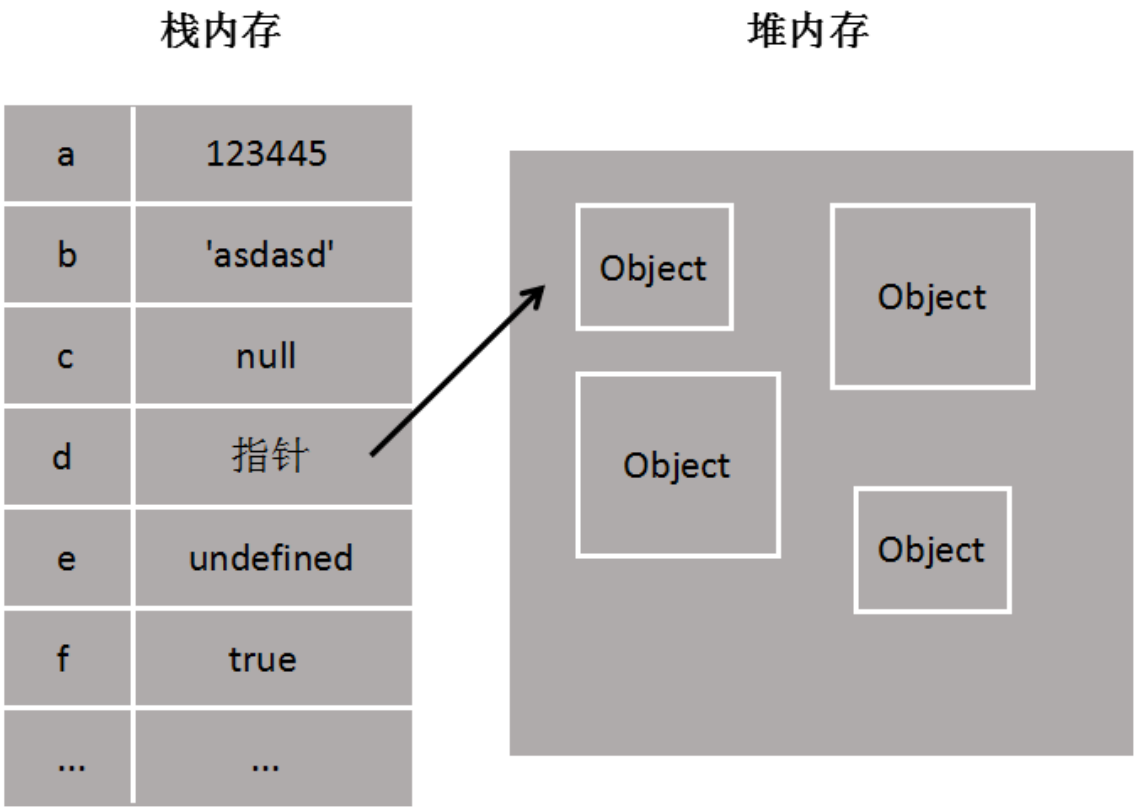
闭包

定义：函数嵌套函数，内部函数就是闭包

- 作用域链(重点理解):向上查找当前需要的变量或函数

利用闭包实现模块化开发，便于管理

堆栈内存概念



- 基本(原始)类型存在栈内存中，栈 stack 为自动分配的内存空间，它由系统自动释放；
- 引用类型存在堆内存中， 堆 heap 是动态分配的内存，大小不定也不会自动释放。

立即执行 a 函数

```
//立即执行函数：函数定义完，立即被调用，这种函数叫做立即执行函数，立即执行函数往往只会执行一次
(function fun a(){})()
```

防抖和节流

- debounce： 只执行最后一次
- throttle： 控制执行次数

客户端渲染和服务端渲染

```
//传统的SPA模式,即客户端渲染的模式
Vue.js构建的应用程序，默认情况下是有一个html模板页，然后通过webpack打包生成一堆js、css等等资源文件。然后塞到index.html中.
用户输入url访问页面 -> 先得到一个html模板页 -> 然后通过异步请求服务端数据 -> 得到服务端的数据 -> 渲染成局部页面 -> 用户

//SSR模式,即服务端渲染模式
用户输入url访问页面 -> 服务端接收到请求 -> 将对应请求的数据渲染完一个网页 -> 返回给用户
```

toString()

- 当对 undefined、null 使用 tostring(),会报错

JavaScript的许多内置对象都重写了该函数，以实现更适合自身的功能需要。

类型	行为描述
String	返回 String 对象的值。
Number	返回 Number 的字符串表示。
Boolean	如果布尔值是true，则返回"true"。否则返回"false"。
Object (默认)	返回"[object ObjectName]"，其中 ObjectName 是对象类型的名称。
Array	将 Array 的每个元素转换为字符串，并将它们依次连接起来，两个元素之间用英文逗号作为分隔符进行拼接。
Date	返回日期的文本表示。
Error	返回一个包含相关错误信息的字符串。
Function	返回如下格式的字符串，其中 functionname 是一个函数的名称 此函数的 toString 方法被调用： “function functionname() { [native code] }”

原型对象

- 每个对象都有其原型对象 `__proto__`
- 构造函数 prototype 属性
- 基于原型的继承

```
function User(name) {
  this.name = name;
  this.login = function () {
    console.log("我登陆了");
  };
}
function Admin(name) {
  this.name = name;
}
//让Admin的原型指向User的实例，实现Admin继承User
Admin.prototype = new User();
```

- ==原型链==

```

```

```
//类class
class Cat{

    constructor(name.age){

    }

}
```

- 类的继承

```
class Admin extend User{
   static xx = '' //静态属性
   constructor(name,age){
        super()
    }
   login(){

   }
}
```

JIT-即时编译

```
JavaScript是一门解释型语言，使用了JIT技术，使得运行速度得到改善
```

js 预编译

```
//https://www.bilibili.com/video/BV1sN411974w?p=6
```

```
// 1. 创建ao对象 AO{}
// 2. 找形参和变量声明 将变量和形参名 当做 ao对象的属性名 值为undefined
// 3. 实参形参相统一
// 4. 在函数体里面找函数声明 值赋予函数体

// AO{
//    a:undefined 1 function a() { }
//    c:undefined 2  function c() { }
//    d:undefined
//    b:undefined
// }
```

lodashJS 库使用

```
//https://blog.csdn.net/weixin_41229588/article/details/106334552

// 以 `user` 升序排序 再  `age` 以降序排序。
//_.orderBy(users, ['user', 'age'], ['asc', 'desc']);

//随机返回元素
//_.sample([1, 2, 3, 4]);
// => 2

//随机返回n个元素
//_.sampleSize([1, 2, 3], 2);
// => [3, 1]
```

defineProperty()开启数据代理

```
let obj = {}

Object.defineProperty(obj,'age',{
    configurable:false,
    enumerable:true,
    writable:true,
    value:18,
    get(){
        return xx;
    }
    set(value){
    ...
    }
})
```

Worker 实现 js 多线程

```
new Worker() 开辟出一个子线程，只能下载网络文件，不能读取本地文件
```

Promise

```
// 在promise中定义好异步
cancel() {
      // promise
      return Promise.resolve(
        new Promise((resolve) => {
          setTimeout(() => {
            ....
            console.log(1)
            resolve();
          }, 0);
        })
      );
},
// 在fn中同步使用
async fn(){
    await this.cancel()
    console.log(2)
}
```

打印结果：先1后2，同步执行异步函数

其他

1.history.scrollRestoration 取消浏览器对页面滚动条位置的记录

使用很简单，在页面的任意位置执行下面几行 JS 代码就可以了：

```
if (history.scrollRestoration) {
  history.scrollRestoration = 'manual';
}
```

TypeScript

类型声明

```
let s: string;
let a: number;
let c: boolean;
function(a: string,b: string): string{
    return a+b;
}

let m = 100; // 直接定义变量并赋值的，不需要再声明变量类型

// 可以使用 | 来连接多个类型、值 (联合类型)
let a: string | number;
let a = 'male' | 'female'

// any表示任意类型，表示关闭该变量ts的类型检测
let d: any;

// unknown表示未知类型,实际就是有类型安全性的any
let e: unknown;
let s = e as string;
let ss = <string>e;

// void 用来表示函数没有返回值
function fn(): void{

}

// never 用来表示函数永远不会返回结果
function fn(): never{
    throw new Error('报错了')
}

// object: {}用于指定对象中可以包含哪些属性
```

```
let a: {namecv :string,age?: number, [propName: string]: any} // ?表示该属性可选，[propName:string]: any表示任意类型任意数量的属性

// 设置函数结构的类型声明
// 语法:(形参：类型，形参：类型)=>返回值类型
let d: (a: number,b: number)=>number

// number[] 表示数值数组
let a: number[]
a = [1,2,3,4,5]
// Array<number> 表示数值数组，泛型
let b: Array<number>
b = [1,2,3,4,5,6]

// enum枚举
enum Gender{
    Male=0,
    Female=1
}
let i: {name: string,gender: Gender}
i = {
    name:'wxx',
    gender: Gender.Male
}

// type:创建类型别名
type myType = 1 | 2 | 3 | 4;
let a: myType;
let b: myType;
```

命令

```
tsc xx.ts // 编译ts文件
tsc xx.ts -w // 开启监听
```

抽象类

```
// abstract声明抽象类，抽象类不能创建实例，只能用来子类继承
abstract class Animal{
   name: string;
   constructor(name: string){
       this.name = name;
   }
  // abstract定义抽象方法，抽象方法只能定义在抽象类中，并且子类必须实现该方法
   abstract sayHello(): void
}

class Cat extends Animal{
    // 子类必须实现父类的抽象方法
    sayHello(){
        console.log('hello')
    }
}
```

接口

```
// interface定义 类的结构
interface obj {
  name: string;
  age: number;
}

interface myInter {
  name: string;
  sayHello(): void;
}
class MyClass implements myInter {
  name: string;
  constructor(name: string) {
    this.name = name;
  }
  sayHello() {
    console.log("hello");
  }
}
```

属性封装

```
// private定义私有属性，只能在MyClass中访问
class MyClass{
    private _name: string;
    private _age: number;
    constructor(name: string){
        this.name = name;
    }
    sayHello(){
        console.log('hello')
    }
    get name(){
        return this._name;
    }
    set name(value){
        this._name = value;
    }
}

// publish 定义的属性可以任意修改

// protected 定义保护属性，该属性可以在当前类和子类中被访问
```

vue

$.set()、Vue.set()

```
//向vue实例动态添加响应式数据，
this.$set(this.user, "sex", "男"); //this指vue实例vm
Vue.set(this.user, "sex", "男"); //

//set(target,key,value),target不能是this(vm)本身
Vue.set(this, "sex", "男"); //错误，key不能是Vue实例的根数据对象
```

监视数组变更原理

```
//vue将被侦听的数组的变更方法进行了包裹
//vue的响应式只能监听数组通过push、pop、reverse、shift、unshfit等方法操作后的变化
arr.splice(0, 0, "w"); //页面发生响应式
//通过直接操作数组索引修改元素的，vue响应式不发生变化
arr[0] = "w"; //页面不发生响应式
```

v-model 修饰符

```
v - model.number; //强制类型转换
v - model.lazy; //懒\响应，当失去焦点时响应
v - model.trim; //去除首尾空字串
```

v-on 修饰符

```
<!-- 阻止单击事件继续传播 -->
<a v-on:click.stop="doThis"></a>

<!-- 提交事件不再重载页面 -->
<form v-on:submit.prevent="onSubmit"></form>

<!-- 修饰符可以串联 -->
<a v-on:click.stop.prevent="doThat"></a>

<!-- 只有修饰符 -->
<form v-on:submit.prevent></form>

<!-- 添加事件监听器时使用事件捕获模式 -->
<!-- 即内部元素触发的事件先在此处理，然后才交由内部元素进行处理 -->
<div v-on:click.capture="doThis">...</div>

<!-- 只当在 event.target 是当前元素自身时触发处理函数 -->
<!-- 即事件不是从内部元素触发的 -->
<div v-on:click.self="doThat">...</div>

<!-- 表示事件只触发一次 -->
<div v-on:click.once="doThat">...</div>
```

过滤器

```
// 过滤器可以连用 比如同时使用timeFormat myslice
{{time | timeFormat | myslice}}
filters:{
   //opt_param 可选参数
   timeFormat(value,opt_param){
        return 'YYYY-MM-DD'
   },
   myslice(value,opt_param){
        return value.split('-')
   }
}
```

v-text

不建议使用;

v-html

```
//注意不法输入，造成xss攻击(冒充用户之手)
```

v-cloak

```
//隐藏vue模板html元素
[v-cloak]{
    display:none;
}
```

v-once

```
<h2 v-once>{{ count }}</h2>
//v-once所在节点在初次动态渲染后就视为静态内容了
//以后数据的改变不会引起v-once所在节点的更新
```

v-pre

```
//使节点跳过编译过程,优化性能
//在不需要编译的html节点上才能使用
```

@keyup.enter

```
//enter抬起时触发
//@keyup 主要针对表单元素
```

自定义指令

```
//定义v-big指令
directives:{
    big(element,binding){
        element.innerText = binding.value*10
    },
    fbig:{
        //指令与页面成功绑定时
        bind(element,binding){},
        //指令所在元素被插入页面时
        inserted(element,binding){},
        //指令所在的模块被重新解析时
        update(element,binding){

     }
}
```

生命周期

```
//注意：有的单词末尾有‘ed’，不要写错
beforeCreate(){}
created(){}
beforeMount(){}  //vue挂在dom前
//vue完成模块的解析并把初始的DOM元素放入页面后（挂载完毕）后执行，只会走一次
mounted(){}
beforeUpdate(){} //更新前
updated(){}   //更新时
beforeDestroy(){}  //销毁前
destroyed(){}  //销毁时
```

创建非单文件组件

```
//Vue.entend创建user组件
let user = Vue.entend({
    template:`<div>
                <h2>hah</h2>
                <h2>hah</h2>
            </div>`,
    data(){
        user:'w',
        age:18,
    },
})
```

注册组件

```
//注册局部组件
components:{
    user,
    ...
}
//注册全局组件
Vue.component('user',user)
```

@click 传参当前 DOM 元素

```
//$event 当前的DOM元素
@click="updateBed(index,$event)"
```

ref

```
<h1 ref="title"></h1>
<School ref="school" />

console.log(this.$refs.title) //DOM对象 console.log(this.$refs.school) //School组件实例对象
```

props 配置项

```
export default {
  //
  props: ["name", "age"],
  //对接收的数据进行类型限制
  props: {
    name: String,
    age: Number,
  },
  props: {
    name: {
      type: String,
      required: true, //必需值
    },
    age: {
      type: Number,
      default: 99, //设置默认值
    },
  },
};
//props接受父组件传过来的值，并且不可以进行修改
```

mixin 混合

```
//mixin混合:将可以复用的配置项提取成混合对象（_mixin）,当前组件和混合配置项重复的配置项，优先使用当前组件的配置项 ，如示例中优先使用 x=999,

//在mixin.js中
export const _mixin = {
    data(){
        return {
            x:666
        }
    }
    methods:{
        showName(){
            alert(this.name)
        }
    },
    mounted(){
        consoloe.log('你好啊')
    }
}

//在组件中使用混合
import {_mixin} from './mixin.js'
export default{
    data(){
```

```
        return {
            x:999
        }
    },
    mixins:[_mixin],//在组件中使用混合
}

//使用全局混合
Vue.mixin(_mixin)
```

组件绑定自定义事件

```
//为Stydent组件绑定getName自定义事件
<Student @getName="getName"></Student>
```

组件绑定原生事件 click

```
//通过native 为Student组件绑定原生的click事件
<Student @click.native="getName"></Student>
```

解除绑定事件

```
this.$off("getData"); //解除getData绑定事件
this.$off(); //解除所有的自定义事件
```

销毁组件

```
this.$destroy(); //销毁当前组件
```

全局事件总线

```
//$bus

new Vue({
    ....
    beforeCreate(){
    Vue.prototype.$bus = this //安装全局事件总线，$bus就是当前应用的vm
}
})

//使用
//1.接受数据，给$bus绑定自定义事件，绑定回调
mounted(){
    this.$bus.$on('xxx',()=>{
        ....
    })
}
//2.提供数据，触发自定义事件，执行回调函数
this.$bus.$emit('xxx',this.demo) //this.demo:要提供的数据，就是参数
```

pubsub-js 订阅发布消息

```
//发布消息
PubSub.publish("hello", "hello world!");

//订阅消息
var mySubscriber = function (msg, data) {
  console.log(msg, data); //hello, hello world!
};
var token = PubSub.subscribe("hello", mySubscriber);

//组件销毁时销毁订阅
PubSub.unsubscribe(token);
```

$nextTick()

```
//在下一次轮次执行回调函数
this.$nextTick(() => {
  alert("xx");
});
```

过渡动画

```
<transition>
</transition>

//transition-group多元素过渡
<transition-group>
    <h1 key="1"></h1>
    <h1 key="2"></h1>
</transition-group>
```

vue 使用插件

```
Vue.use(XXX); //
```

slot 插槽

```
//1.定义默认插槽，由组件的使用者进行填充
<slot></slot>



//2.具名插槽 ，在子组件中定义
//https://www.bilibili.com/video/BV1Zy4y1K7SH?p=103&spm_id_from=pageDriver
<slot name="wxx"></slot>

//在父组件中使用
<a slot="wxx" src="http://baidu.com"></a>
或
<template v-slot:wxx>
    ...
</template>
或
<template #wxx>
```

```
    ...
</template>


//3.作用域插槽
//在父组件中使用子组件，并且可以使用子组件中的值
// 子组件
<slot name="wxx" :value="child"></slot>
data(){
    child:'haha'
}

// 父组件
//在父组件中使用，通过v-slot:wxx="obj"的方式将子组件值赋值给obj
<template v-slot:wxx="obj">
    {{obj.value}} // 显示'haha'
</template>
```

router-link

```
//router-link 最终转化为a标签
<router-link active-class="xxx" to="/home">
  home
</router-link>
```

vue3学习

```
// 基于setup语法糖的常用语法
// defineProps是vue3提供的方法，不需要引用，可以直接使用;props的数据都是只读的，子组件中不能修改
let props = defineProps(['time','name'])
```

uniapp 学习

简单学习

```
// 简单学习：类vue语法
1.敲clog：可直接输出console.log()；敲clogv：可输出console.log(": " + );，并且出现双光标，方便把变量名称和值同时打印出来
```

```

```

GIS 学习

WFS

WebGIS服务器除了能返回一张地图图像之外，也可以返回绘制该地图图像所使用的真实地理数据。用户利用这些数据可以创建他们自己的地图与应用、数据格式转换以及底层的地理操作。这类返回地理要素数据的规范称为WFS（Web Feature Service--Web要素服务）。

只要服务器和客户端遵循统一规范，那么服务器与客户端传送的数据可以是任意格式的。为了规范通过Web服务发送矢量数据的过程，OGC制定了WFS规范。

WFS 与 WMS 区别

WMS是由服务器将一地图图像发送给客户端，而WFS是服务器将矢量数据发送给客户端

==也就是在使用WMS时地图由服务器绘制，在使用WFS时地图由客户端绘制。==

WMTS 和 WMS 区别

// WMTS
WMTS提供了一种采用预定义图块方法发布数字地图服务的标准化解决方案。WMTS弥补了WMS不能提供分块地图的不足。WMS针对提供可定制地图的服务，是一个动态数据或用户定制地图（需结合SLD标准）的理想解决办法。WMTS牺牲了提供定制地图的灵活性，代之以通过提供静态数据（基础地图）来增强伸缩性，这些静态数据的范围框和比例尺被限定在各个图块内。

//WMS
Web Map Service，网络地图服务，它是利用具有地理空间位置信息的数据制作地图，其中将地图定义为地理数据的可视化表现，能够根据用户的请求，返回相应的地图，包括PNG、GIF、JPEG等栅格形式，或者SVG或者WEB CGM等矢量形式。WMS支持HTTP协议，所支持的操作是由URL决定的。可以通过SLD定制地图

ol 添加非默认控件

```
var map = new ol.Map({
  //将DragRotateAndZoom控件添加到地图
  interactions: ol.interaction.defaults().extend([new
ol.interaction.DragRotateAndZoom()]),
  layers: [
    new ol.layer.Tile({
      source: new ol.source.OSM(),
    }),
  ],
  target: "map",
  view: new ol.View({
    center: [0, 0],
    zoom: 2,
  }),
});
```

ol 中 DOM 元素的组织关系

如图:

当调用ol.Map()这个构造函数时，OpenLayers地图引擎会在内部创建一个视口容器（viewport container，一个css类名为ol-viewport的div DOM元素）
并将其放置在target属性映射的地图容器元素中。

而在视口容器中将会包含三个子元素：

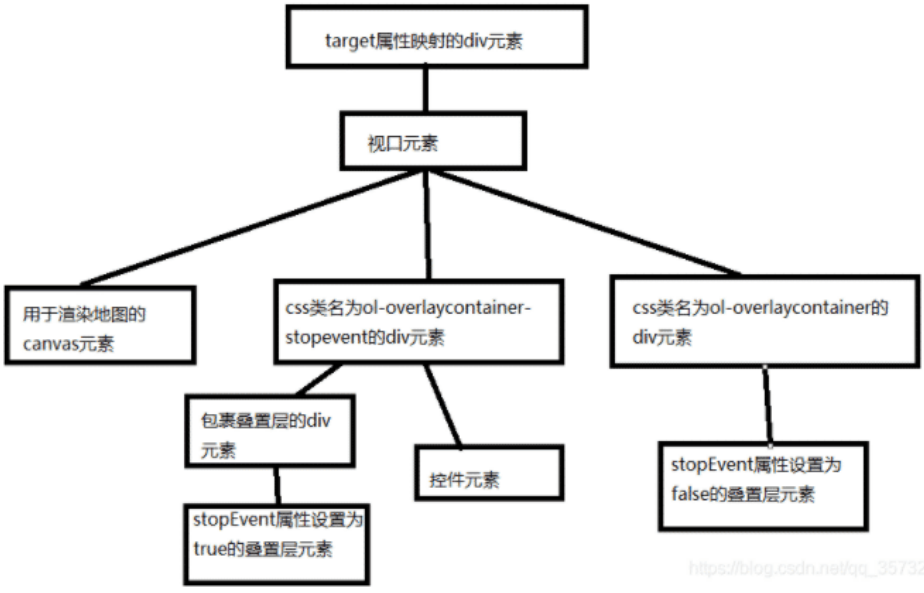
canvas元素 -- 用于渲染地图

css类名为ol-overlaycontainer-stopevent的div元素 -- 用于承载控件（control）和stopEvent属性设置为true的叠置层（overlay），此处的DOM元素事件不冒泡

css类名为ol-overlaycontainer -- 用于承载stopEvent属性设置为false的叠置层，此处的DOM元素事件会冒泡

所以上面示例中用于充当叠置层的html元素都会被移到用于承载叠置层的div元素中。

所以OpenLayers的DOM元素组织结构为：



cesium

投影支持

Cesium只支持WCS84和墨卡托投影；
其中地形默认为WCS84投影，且一般只支持WCS84投影；
影像数据两种投影都支持，默认为墨卡托投影，WebMercatorTilingScheme平铺方案
如果影像数据为墨卡托投影，则cesium内部会自动进行动态投影矫正，会有一定的计算量，但是不需要额外进行代码编写；
当影像数据为WCS84时，需要手动申明该ImageryProvider的TilingScheme=new Cesium.GeographicTilingScheme()

加载B3DM优化建议

```
// 原文：https://blog.csdn.net/weixin_42539678/article/details/122683572
var tileset = new Cesium.Cesium3DTileset({
        url:url,
        skipLevelOfDetail: true,
        baseScreenSpaceError: 1024,
        maximumScreenSpaceError: 256, // 数值加大，能让最终成像变模糊
        skipScreenSpaceErrorFactor: 16,
        skipLevels: 1,
        immediatelyLoadDesiredLevelOfDetail: false,
        loadSiblings: true, // 如果为true则不会在已加载完概况房屋后，自动从中心开始超清化房屋
        cullWithChildrenBounds: true,
        cullRequestsWhileMoving: true,
        cullRequestsWhileMovingMultiplier: 10, // 值越小能够更快的剔除
        preloadWhenHidden: true,
        preferLeaves: true,
        maximumMemoryUsage: 128, // 内存分配变小有利于倾斜摄影数据回收，提升性能体验
        progressiveResolutionHeightFraction: 0.5, // 数值偏于0能够让初始加载变得模糊
        dynamicScreenSpaceErrorDensity: 0.5, // 数值加大，能让周边加载变快
        dynamicScreenSpaceErrorFactor: 1, // 不知道起了什么作用没，反正放着吧先
        dynamicScreenSpaceError: true, // 根据测试，有了这个后，会在真正的全屏加载完之后才清晰化房屋
})
```

主要类及方法总结

```
js:
1.Cesium.Matrix4.IDENTITY 一个常量，代表单位矩阵，任何矩阵*单位矩阵=本身
2.viewer.entities.removeAll() 删除viewer上所有的实体entity
3.HeadingPitchRoll 设置对象的水平方向、俯仰角度和翻滚角度
4.scene.clampToHeightSupported 判断计算机是否支持根据模型或者地形等物体计算点位的附着高度
5.Cesium.Cartesian3.lerp(a,b,c) a~b之间线性计算出在c处的值
6.scene.clampToHeightMostDetailed(cartesians)  在cartesians坐标处计算出对应的附着高度，可能是附着在地形、entity、primitives、3dtiles等上
7.scene.camera.setView 设置相机的位置、方向和变换矩阵
  scene.camera.lookAt()
8.PolylineOutlineMaterialProperty 用来描述线轮廓的材质
9.PolylineGraphics.depthFailMaterial 用于指定线被深度检测盖住的部分的材质
10.设置viewer.sceneMode和viewer.mapMode2D可以使用2D和2.5地图。
11.viewer.resolutionScale 设置cesium的呈现分辨率的缩放系数
12.d=Cesium.Math.clamp(a,b,c) 用于将c值限定在a~b之间，当c<a时，d=a当c>b时，d=b;否则d=c
13.Cesium.Color.fromCssColorString() 将CSS颜色值转化为Cesium.Color
14.Cesium.ClassificationType 设置对象是否影响地形，3dtiles或两者。
15.viewer.projectionPick 设置相机的投影方式（正射投影或者透视投影），透视投影更符合人类眼睛的观察模式
```

16.viewer.projectionPicker.viewModel.switchToPerspective() 手动设置相机切换到透视投影
17.Cesium.Transforms.headingPitchRollQuaternion() 根据提供的原点为中心，进行欧拉角计算，得到一个新的四元数，在设置entity的坐标和方向时用到
18.viewer.trackedEntity=entity 设置相机跟踪当前entity
19.PostProcessStageLibrary 常见的后处理函数
20.Cesium.Cartesian3.fromDegreesArray()
   Cesium.Cartesian3.fromDegreesArrayHeights
21.PolylineDashMaterialProperty 定义虚线材质
22.PolylineGlowMaterialProperty 定义发光线材质
23.PolylineArrowMaterialProperty 定义线箭头材质
24.cornerType: Cesium.CornerType.BEVELED 设置图形拐角处的连接类型
25.ploygon.perPositionHeight 设置polygon是否使用每个位置的高度，仅当extrudedHeight=0时，设置ploygon.perPositionHeight=true才生效
26.polygon 可以构造多边形以及多边形体，灵活应用extrudedHeight和perPositionHeight
27.viewer.entities.add() 添加实体
28.viewer.zoomTo()
29.世界坐标转化经纬度：
```
    cartographic = Cesium.Cartographic.fromCartesian(cartesian)
    lon = Cesium.Math.toDegrees(cartographic.longitude)
    lat = Cesium.Math.toDegrees(cartographic.latitude)
```
30.Cesium.Color.clone() 颜色拷贝
31.Cesium.Color.GREEN.withAlpha(0.5) 带透明度的颜色
32.handler = new Cesium.ScreenSpaceEventHandler(scene.canvas) 定义用户输入事件
   handler.setInputAction(fn,type, modifier) 开启type类型输入事件，并设置执行的功能函数
   handler.destroy() 销毁用户输入事件监听
33.scene.drillPick(movement.endPosition) 根据屏幕坐标深度捕捉图元对象，获得一个对象数组
34.viewer.scene.camera.pickEllipsoid(cartesian2) 根据屏幕坐标计算世界坐标
35.Cesium.Cartesian3.distance(a,b) 计算ab两点间的距离
36.viewer.camera.changed.addEventListener(fn) 开启相机事件监听
   viewer.camera.percentageChanged = 0.1 让相机变得更加灵敏
37.viewer.scene.cartesianToCanvasCoordinates(cartesia3,cartesia2) 将世界坐标映射为屏幕坐标
38.关于entity中graphics的分类：
   billboard 广告牌
   box 构造立方体，中心位置和方向由包含的 Entity 确定
   Corridor 构造走廊，可以挤压成体积
   Cylinder 构造圆柱体、截锥体或者圆锥体，中心位置和方向由包含的 Entity 确定
   Ellipse 构造由长短轴定义的椭圆，可以挤压成体积,中心位置和方向由包含的 Entity 确定
   Ellipsoid 构造椭球体或者球体,中心位置和方向由包含的 Entity 确定
   Label 构造二维标签,中心位置和方向由包含的 Entity 确定
   model 构造基于gltf的模型,中心位置和方向由包含的 Entity 确定
   path 构造entity随时间移动时形成的路径的折线
   plane 构造只有长宽的平面，需要设置其法线属性
   point 构造图形点
   polygon 构造外部形状和任何嵌套孔的线性环的层次结构定义的多边形，可以挤压体积
   polyline 构造线段
   PolylineVolume 构造折线体积，管线等...
   Rectangle 构造矩形，可以挤压成体积
   Wall 构造二维墙体
39.Cesium.HeightReference 设置相对于地形的位置
40.viewer.dataSources.raise(dataSource) 将 dataSource的显示z-index上升一位
41.Cesium.Camera.·· = Cesium.Rectangle.fromDegrees(73.0, 3.0, 135.0, 53.0);//设置Home位置
   viewer.camera.flyHome(5);
42.Cesium内置了拦截数据相应的方法：如下监听了时钟速率发现变化的事件
```
  Cesium.knockout
      .getObservable(viewer.clockViewModel, "multiplier")
      .subscribe(function (newValue) {
```

```
        console.log(newValue,110);
      });
```

43.FXAA和MSAA

FXAA:只是一个后处理技术，对原图绘制完成后，通过算法识别边缘，然后以像素级别进行混合；

MSAA:MSAA中每个像素点有4个子采样点，每个三角形对每个像素点只在中心点着色1次，再把计算结果根据深度和覆盖信息保存到对应的子采　　样点，最后对4个采样点取均值作为最终的像素颜色

44.后处理中PostProcessStageCollection和PostProcessStage、PostProcessStageComposite、PostProcessStageLibrary的关系：

PostProcessStageCollection最大，是PostProcessStage或者PostProcessStageComposite的集合，　viewer.scene.postProcessStages.add()将后处理应用至场景中；

PostProcessStageComposite是多个PostProcessStage的集合，在执行逻辑上是同时进行的；

PostProcessStageLibrary是用于创建通用的PostProcessStage的函数，即事先cesium定义好了的PostProcessStage,不需要自定义　fragmentshader；

flyto

```
//xxx 一般为3dtidle和model等实体对象
viewer.flyto(xxx);

//xxx 一般为明确的坐标地点destination，还需要设置orientation方位角
camera.flyto(xxx);
```

clampToHeight

```
//返回cartesian位置处objectsToExclude上的夹紧位置，即紧贴实体的位置
clampToHeight(cartesian, objectsToExclude); //objectsToExclude 一般为实体或3DTiles
```

sampleTerrainMostDetailed、sampleTerrain

```
//在terrain数据集的最大可用图块级别上获得高程
Cesium.sampleTerrainMostDetailed(terrainProvider, positions);

//在terrain数据集的level级别上获得高程
Cesium.sampleTerrain(terrainProvider, level, positions);
```

项目实战

```
1.工具栏
2.基础数据展示：
  行政区划
  建筑
  道路
  地铁
  兴趣点
  模型
  地下管网
3.
```

webgl

model、view 矩阵分析

//https://zhuanlan.zhihu.com/p/34672417
正如把大象装进冰箱一样，把3D世界中的物体呈现在我们人眼所见的屏幕上也需要三步：把物体放入世界中，将摄像头指向物体，最后把摄像头所观察到的事物投影到屏幕。这三个环节的执行是通过将物体的齐次坐标与不同阶段所需的矩阵相乘得到的.

总结：通过观察矩阵，我们把物体转移到了摄像头面前，之后旋转摄像头视角，准备把视角范围内的物体通过投影矩阵映射到屏幕。

基本绘制步骤

不管怎么说吧，基础是很重要的。那就在这些基础上，来绘制一个多边形吧。需要
先，来确认一下绘制的步骤。

- 从HTML中获取canvas对象
- 从canvas中获取WebGL的context
- 编译着色器
- 准备模型数据
- 顶点缓存（VBO）的生成和通知
- 坐标变换矩阵的生成和通知
- 发出绘图命令
- 更新canvas并渲染

Python

GIT 基本使用

解决冲突

Git会在发生冲突的地方修改文件的内容，如下图。所以我们需要手动修正冲突。

如上图：

==分割线上方是本地数据库的内容，
下方是远程数据库的编辑内容。

<<<<<<<<<
    发生冲突的部分
>>>>>>>>>

TortoiesGit 的使用

1.https://backlog.com/git-tutorial/cn/intro/intro1_2.html
2.https://blog.csdn.net/u011966339/article/details/106250920

软知识

传参

```
// request payload
'content-type': 'application/json; charset=utf-8'

//form data
'Content-Type': 'application/x-www-form-urlencoded;charset=UTF-8'
```

插件和 SDK 的区别

//以cesium为例
插件：就是自己基于cesium的某个部分（比如Layer或者entity）做的二次开发。最终项目里面不包括cesium源码，他的运行和使用需要自己导入cesium作为依赖。

SDK：也是基于cesium做的维持开发，但最终项目包括了自己开发的代码和cesium的源码以及其他依赖插件和环境等。

常用工具集合

1.popmotion.js 取代锚点定位的好工具，滚动到指定区域。
2.https://shields.io/category/social 获得github项目实时的星星数
3.vxe-table 强大的vue 表格插件
4.npm install coordtransform 一个提供了百度坐标（BD09）、国测局坐标（火星坐标，GCJ02）、和WGS84坐标系之间的转换的工具模块
5.npm install vue-quill-editor --save vue富文本编辑器的使用