

前端学科面试宝典

前端学科面试宝典

目录

1、H5 的新特性有哪些？C3 的新特性有哪些？（必会）	11
2、如何使一个盒子水平垂直居中？（必会）	13
3、如何实现双飞翼（圣杯）布局？（必会）	18
1、利用定位实现两侧固定中间自适应.....	18
2、利用 flex 布局实现两侧固定中间自适应.....	19
3、利用 bfc 块级格式化上下文, 实现两侧固定中间自适应.....	21
4、CSS 的盒模型？（必会）	23
5、CSS 中选择器的优先级以及 CSS 权重如何计算？（必会）	24
6、列举 5 个以上的 H5input 元素 type 属性值？（必会）	26
7、CSS 中哪些属性可继承，哪些不可以？（必会）	27
8、CSS 单位中 px、em 和 rem 的区别？（必会）	28
9、rem 适配方法如何计算 HTML 跟字号及适配方案？（必会）	29
10、Display: none 与 visibility: hidden 的区别？（必会）	30
11、Position 的值有哪些，分别有哪些作用？（必会）	31
12、为什么会出现浮动？浮动元素会引起什么问题？如何清除浮动？（必会）	32
13、简述弹性盒子 flex 布局及 rem 布局？（必会）	33
14、如何解决 margin “塌陷” ？（必会）	39
15、less 和 Scss 的配置使用以及特点？（必会）	40
1、可以相互调用，但是不能拿自己调用自己，形成递归.....	42
2、通过@include 引用.....	42

16、::before 和::after 中双冒号和单冒号有什么区别、作用？（必会）	42
17、CSS3 新增伪类，以及伪元素？（必会）	43
18、Bootstrap 栅格系统的工作原理？（必会）	44
19、自己手动封装响应式布局方案？（必会）	47
20、BFC 是什么？（高薪常问）	49
21、什么是渐进增强和优雅降级？它们有什么不同？（了解）	51
22、iframe 有哪些优缺点？（了解）	52
23、使用 CSS 怎么让 Chrome 支持小于 12px 的文字比如 10px？（了解）	53
1、JavaScript 的基本类型有哪些？引用类型有哪些？null 和 undefined 的区别？（必会）	54
2、如何判断 JavaScript 的数据类型？（必会） JavaScript 数据类型一共有 7 种：	55
3、简述创建函数的几种方式？（必会）	58
4、Javascript 创建对象的几种方式？（必会）	59
5、请指出 JavaScript 宿主对象和原生对象的区别？（必会）	61
6、JavaScript 内置的常用对象有哪些？并列举该对象常用的方法？（必会）	62
7、=== 和 ==的区别？（必会）	70
8、null, undefined 的区别（必会）	71
9、JavaScript 中什么情况下会返回 undefined 值？（必会）	72
10、如何区分数组 and 对象？（必会）	72
11、多维数组降维的几种方法（必会）	73
12、怎么判断两个对象相等？（必会）	75
13、列举三种强制类型转换和两种隐式类型转换？（必会）	76

14、JavaScript 中怎么获取当前日期的月份？（必会）	77
15、什么是类数组（伪数组）， 如何将其转化为真实的数组？（必会）	78
16、如何遍历对象的属性？（必会）	79
17、如何使用原生 JavaScript 给一个按钮绑定两个 onclick 事件？（必会）	81
18、JavaScript 中的作用域、预解析与变量声明提升？（必会）	82
19、变量提升与函数提升的区别？（必会）	85
20、什么是作用域链？（必会）	85
21、如何延长作用域链？（必会）	86
22、判断一个值是什么类型有哪些方法？（必会）	86
23、如何实现数组的随机排序？（必会）	87
24、src 和 href 的区别是？（了解）	88
1、 什么是 dom？（必会）	89
2、 dom 节点的 Attribute 和 Property 有何区别？（必会）	90
3、 dom 结构操作怎样添加、移除、移动、复制、创建和查找节点？（必会）	91
1、创建新节点.....	91
2、添加、移除、替换、插入.....	91
3、查找.....	91
4、 dom 事件模型？（必会）	91
5、什么是事件冒泡，它是如何工作的？如何阻止事件冒泡、默认行为？（必会）	92
6、JavaScript 动画和 CSS3 动画有什么区别？（必会）	94
7、 event 对象的常见应用？（必会）	96

8、通用事件绑定/ 编写一个通用的事件监听函数? (必会)	96
9、DOM 和 BOM 的区别 (必会)	97
10、事件三要素 (必会)	98
11、事件执行过程 (必会)	98
12、获取元素位置 (必会)	99
13、封装运动函数 (必会)	100
14、绑定事件和解除事件的区别 (必会)	101
15、谈谈事件委托的理解? (必会)	102
16、JavaScript 中的定时器有哪些? 他们的区别及用法是什么? (必会)	103
17、比较 attachEvent 和 addEventListener? (必会)	103
18、document.write 和 innerHTML 的区别? (必会)	105
19、什么是 window 对象? 什么是 document 对象? (必会)	105
20、Js 拖动的原理? (必会)	109
21、描述浏览器的渲染过程, DOM 树和渲染树的区别 (必会)	110
22、如何最小化重绘(repaint)和回流(reflow) (必会)	111
23、简单说一下页面重绘和回流? (高薪常问)	112
24、Js 延迟加载的方式有哪些? (了解)	112
25、IE 与标准事件模型有哪些差别? (了解)	116
1.1) 解决: 1.1.1) 创建意外的全局变量	122
1.2) 被忘记的 Timers 或者 callbacks	122
1.3) 闭包: 一个可以访问外部 (封闭) 函数变量的内部函数	123

1.4) DOM 引用.....	123
forEach 和 map 的相同点.....	128
1、echarts 的基本用法（必会）	158
2、如何使用 echarts（必会）	159
3、echarts 如何画图？（必会）	159
4、echarts 绘制条形图（必会）	159
5、切换其他组件统计图时，出现卡顿问题如何解决（必会）	160
6、echarts 图表自适应 div resize 问题（必会）	161
7、echarts 在 vue 中怎么引用？（必会）	161
8、echarts 支持哪些图标？（了解）	163
1、什么是 Ajax，Ajax 的原理，Ajax 都有哪些优点和缺点？（必会）	163
2、常见的 HTTP 状态码以及代表的意义（必会）	164
3、请介绍一下 XMLHttpRequest 对象及常用方法和属性（必会）	166
4、Ajax 的实现流程是怎样的？（必会）	168
5、Ajax 接收到的数据类型有哪些，数据如何处理？（必会）	170
6、请解释一下 JavaScript 的同源策略（必会）	171
7、为什么会有跨域的问题出现，如何解决跨域问题（必会）	171
8、Get 和 Post 的区别以及使用场景（必会）	173
9、解释 jsonp 的原理（必会）	174
10、封装好的 Ajax 里的常见参数及其代表的含义（必会）	175
11、jQuery 中 Ajax 与 fetch 、axios 有什么区别？（必会）	176

12、Ajax 注意事项及适用和不适用场景（必会）	178
13、HTTP 与 HTTPS 的区别（必会）	179
14、LocalStorage、sessionStorage、cookie 的区别（必会）	179
15、简述 web 前端 Cookie 机制，并结合该机制说明会话保持原理？（必会）	180
16、一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么（高薪常问） ...	183
17、你知道的 HTTP 请求方式有几种（高薪常问）	183
1、 ES5 和 ES6 的区别，说几个 ES6 的新增方法（必会）	189
2、ES6 的继承和 ES5 的继承有什么区别（必会）	195
3、var、let、const 之间的区别（必会）	195
4、Class、extends 是什么，有什么作用（必会）	196
<code>i == 9999 && resolve();</code>	208
1、git 的基本使用方法（必会）	212
2、git 工作流程（必会）	213
1、需要合并别人代码进来.....	214
2、需要切换分支.....	214
3、我们如何使用 git 和开源的码云或 github 上面的远端仓库的项目进行工作呢（必会）	215
4、git、github、gitlab 三者之间的联系以及区别（必会）	218
5、github 和码云的区别（必会）	220
6、提交时发生冲突，你能解释冲突是如何产生的吗？你是如何解决的（必会）	221
7、如果本次提交误操作，如何撤销（必会）	222
8、git 修改提交的历史信息（必会）	222

9、如何删除 github 和 gitlab 上的文件夹（必会）	223
10、如何查看分支提交的历史记录？查看某个文件的历史记录呢（必会）	224
11、git 跟 svn 有什么区别（必会）	225
12、我们在本地工程常会修改一些配置文件，这些文件不需要被提交，而我们又不想每次执行 git status 时都让这些文件显示出来，我们该如何操作（必会）	225
13、git fetch 和 git merge 和 git pull 的区别（必会）	226
14、如何把本地仓库的内容推向一个空的远程仓库（高薪常问）	错误！未定义书签。
1、开发背景.....	错误！未定义书签。
2、系统架构.....	错误！未定义书签。
3、登录模块.....	错误！未定义书签。
4、首页模块.....	错误！未定义书签。
5、文章类别管理.....	错误！未定义书签。
6、文章列表管理.....	错误！未定义书签。
7、前台页面文章获取.....	错误！未定义书签。
8、项目介绍话术.....	错误！未定义书签。
1、对 Node.js 有没有了解，它有什么特点，适合做什么业务（必会）	错误！未定义书签。
2、Node 和 前端项目怎么解决跨域的（必会）	错误！未定义书签。
3、Node 的优点是什么？缺点是什么（必会）	错误！未定义书签。
4、commonJS 中的 require/exports 和 ES6 中import/export 的区别是什么（必会）	错误！未定义书签。
5、简述同步和异步的区别，如何避免回调地狱，Node 的异步问题是如何解决的（必会）	错误！未定义书签。
6、dependencies 和 devDependencies 两者区别（必会）	错误！未定义书签。

-
- 7、什么是前后端分离的项目?什么是 JS 渲染的项目, 前端渲染和后端渲染的区别 (高薪常问)
..... 错误! 未定义书签。
- 8、mysql 和 mongoDB 有什么区别 (高薪常问) 错误! 未定义书签。
- 1、基本定义..... 错误! 未定义书签。
- 2、Webpack 的优点是什么? (必会) 错误! 未定义书签。
- 3、Webpack 的构建流程是什么?从读取配置到输出文件这个过程尽量说全 (必会) 错误! 未定义书签。
- 4、说一下 Webpack 的热更新原理(必会)..... 错误! 未定义书签。
- 5、Webpack 与 grunt、gulp 的不同? (必会) 错误! 未定义书签。
- 6、有哪些常见的 Loader? 他们是解决什么问题的? (必会) 错误! 未定义书签。
- 7、Loader 和 Plugin 的不同? (必会) 错误! 未定义书签。
- 8、如何利用 Webpack 来优化前端性能 (高薪常问) 错误! 未定义书签。
- 9、是否写过 Loader 和 Plugin? 描述一下编写 loader 或 plugin 的思路? (高薪常问) 错误! 未定义书签。
- 10、使用 Webpack 开发时, 你用过哪些可以提高效率的插件? (高薪常问) 错误! 未定义书签。
- 11、什么是长缓存? 在 Webpack 中如何做到长缓存优化? (高薪常问) . 错误! 未定义书签。
- 12、如何提高 Webpack 的构建速度? (高薪常问) 错误! 未定义书签。
- 13、怎么实现 Webpack 的按需加载? 什么是神奇注释?(高薪常问)..... 错误! 未定义书签。
- 1、如何进行前端性能优化? (必会) 错误! 未定义书签。
- 2、一个页面上有大量的图片 (大型电商网站), 加载很慢, 你有哪些方法优化这些图片的加载,
给用户更好的体验。 (必会) 错误! 未定义书签。
- e.stopPropagation();//W3C 标准..... 错误! 未定义书签。
- 计算机组成图..... 错误! 未定义书签。

H5 移动 web 开发

1、H5 的新特性有哪些？C3 的新特性有哪些？（必会）

H5 新特性

1、拖拽释放(Drap and drop) API ondrop

拖放是一种常见的特性，即抓取对象以后拖到另一个位置

在 HTML5 中，拖放是标准的一部分，任何元素都能够拖放

2、自定义属性 data-id

3、语义化更好的内容标签(header,nav,footer ,aside, article, section)

4、音频 ,视频(audio, video) 如果浏览器不支持自动播放怎么办?在属性中添加 autoplay

5、画布 Canvas

5.1) getContext() 方法返回一个用于在画布上绘图的环境

Canvas.getContext(contextID) 参数 contextID 指定了您想要在画布上绘制的类型。当

前唯一的合法值是 “2d” ，它指定了二维绘图，并且导致这个方法返回一个环境对象，

该对象导出一个二维绘图 API

5.2) cxt.stroke() 如果没有这一步 线条是不会显示在画布上的

5.3) canvas 和 image 在处理图片的时候有什么区别？

image 是通过对象的形式描述图片的,canvas 通过专门的 API 将图片绘制在画布上.

- 6、地理(Geolocation) API
- 7、本地离线存储 localStorage 长期存储数据 浏览器关闭后数据不丢失
- 8、sessionStorage 的数据在浏览器关闭后自动删除
- 9、表单控件 calendar , date , time , email , url , search , tel , file , number
- 10、新的技术 webworker, websocket , Geolocation

CSS3 新特性

- 1、颜色: 新增 RGBA , HSLA 模式
- 2、文字阴影(text-shadow)
- 3、边框: 圆角(border-radius) 边框阴影 : box-shadow
- 4、盒子模型: box-sizing
- 5、背景:background-size background-origin background-clip
- 6、渐变: linear-gradient , radial-gradient
- 7、过渡 : transition 可实现动画
- 8、自定义动画 animate @keyfrom
- 9、媒体查询 多栏布局 @media screen and (width:800px) {...}
- 10、border-image
- 11、2D 转换:transform: translate(x,y) rotate(x,y) skew(x,y) scale(x,y)
- 12、3D 转换
- 13、字体图标 font-face

2、如何使一个盒子水平垂直居中？（必会）

方法一：利用定位（常用方法,推荐）

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .parent {
      width: 500px;
      height: 500px;
      border: 1px solid #000;
      position: relative;
    }

    .child {
      width: 100px;
      height: 100px;
      border: 1px solid #999;
      position: absolute;
      top: 50%;
      left: 50%;
      margin-top: -50px;
      margin-left: -50px;
    }
  </style>
</head>
<body>
  <div class="parent">
    <div class="child">我是子元素</div>
  </div>
</body>
</html>
```

方法二：利用 `margin:auto`;

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .parent {
      width: 500px;
      height: 500px;
      border: 1px solid #000;
      position: relative;
    }

    .child {
      width: 100px;
      height: 100px;
      border: 1px solid #999;
      position: absolute;
      margin: auto;
      top: 0;
      left: 0;
      right: 0;
      bottom: 0;
    }
  </style>
</head>
<body>
  <div class="parent">
    <div class="child">我是子元素</div>
  </div>
</body>
</html>
```

方法三：利用 `display:table-cell`

```
<!DOCTYPE html>
<html lang="en">

<head>
```

```
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
<style>
    .parent {
        width: 500px;
        height: 500px;
        border: 1px solid #000;
        display: table-cell;
        vertical-align: middle;
        text-align: center;
    }

    .child {
        width: 100px;
        height: 100px;
        border: 1px solid #999;
        display: inline-block;
    }
</style>
</head>
<body>
    <div class="parent">
        <div class="child">我是子元素</div>
    </div>
</body>
</html>
```

方法四：利用 display: flex;设置垂直水平都居中

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .parent {
            width: 500px;
            height: 500px;
            border: 1px solid #000;
            display: flex;
```



```
        justify-content: center;
        align-items: center;
    }

    .child {
        width: 100px;
        height: 100px;
        border: 1px solid #999;
    }
</style>
</head>
<body>
    <div class="parent">
        <div class="child">我是子元素</div>
    </div>
</body>
</html>
```

方法五：计算父盒子与子盒子的空间距离(这跟方法一是一个道理)

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <style>
        .parent {
            width: 500px;
            height: 500px;
            border: 1px solid #000;
        }

        .child {
            width: 100px;
            height: 100px;
            border: 1px solid #999;
            margin-top: 200px;
            margin-left: 200px;
        }
    </style>
</head>
```

```
<body>
  <div class="parent">
    <div class="child">我是子元素</div>
  </div>
</body>
</html>
```

方法六：利用 transform

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .parent {
      width: 500px;
      height: 500px;
      border: 1px solid #000;
      position: relative;
    }

    .child {
      width: 100px;
      height: 100px;
      border: 1px solid #999;
      position: absolute;
      top: 50%;
      left: 50%;
      transform: translate(-50%, -50%);
    }
  </style>
</head>
<body>
  <div class="parent">
    <div class="child">我是子元素</div>
  </div>
</body>
</html>
```

3、如何实现双飞翼（圣杯）布局？（必会）

1、利用定位实现两侧固定中间自适应

1.1) 父盒子设置左右 padding 值

1.2) 给左右盒子的 width 设置父盒子的 padding 值,然后分别定位到 padding 处.

1.3) 中间盒子自适应

具体 CSS 代码：

```
<style>

.father {

    height: 400px;

    background-color: pink;

    position: relative;

    padding: 0 200px;

}

.left,.right {

    width: 200px;

    height: 300px;

    background-color: yellow;

    position: absolute;

    top: 0;

}
```

```
.left {  
  
    left: 0;  
  
}  
  
.right {  
  
    right: 0;  
  
}  
  
.center {  
  
    background-color: blue;  
  
    height: 350px;  
  
}  
  
</style>
```

html 结构

```
<div class="father">  
  
    <div class="left"> </div>  
  
    <div class="center"> </div>  
  
    <div class="right"> </div>  
  
</div>
```

2、利用 flex 布局实现两侧固定中间自适应

2.1) 父盒子设置 display:flex;

2.2) 左右盒子设置固定宽高

2.3) 中间盒子设置 flex:1 ;

```
<style>

.father {

    height: 400px;

    background-color: pink;

    display: flex;

}

.left {

    width: 200px;

    height: 300px;

    background-color: yellow;

}

.right {

    width: 200px;

    height: 300px;

    background-color: yellow;

}

.center {

    flex: 1;

    background-color: blue;

}
```

```
</style>
```

html 结构

```
<div class="father">

    <div class="left"> </div>

    <div class="center"> </div>

    <div class="right"> </div>

</div>
```

3、利用 bfc 块级格式化上下文, 实现两侧固定中间自适应

3.1) 左右固定宽高, 进行浮动

3.2) 中间 overflow: hidden;

```
<style>

    .father {

        height: 500px;

        background-color: pink;

    }

    .left {

        float: left;

        width: 200px;

        height: 400px;

        background-color: blue;
```

```
}

.right {

    float: right;

    width: 200px;

    height: 400px;

    background-color: blue;

}

.center {

    height: 450px;

    background-color: green;

    overflow: hidden;

}

</style>
```

html 结构

```
<!-- 注意:left 和 right 必须放在 center 前面 -->

<div class="father">

    <div class="left"> </div>

    <div class="right"> </div>

    <div class="center"> </div>

</div>
```

4、CSS 的盒模型？（必会）

盒子模型（Box Model）可以用来对元素进行布局，包括内边距，边框，外边距，和实际内容这几个部分

盒子模型分为两种：

第一种是 W3C 标准的盒子模型（标准盒模型）

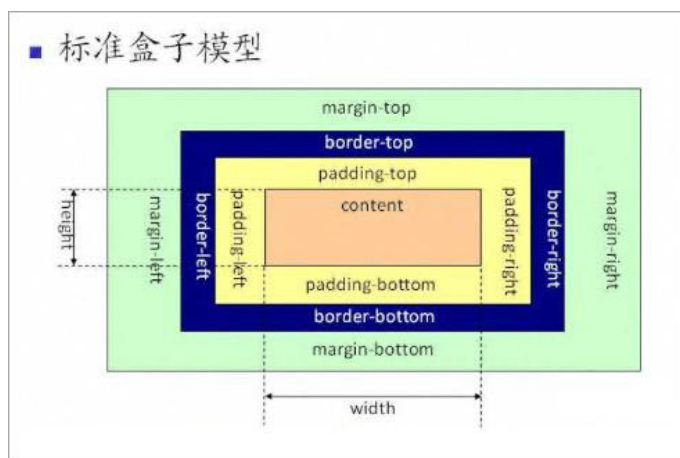
第二种 IE 标准的盒子模型（怪异盒模型）

标准盒模型与怪异盒模型的表现效果的区别之处：

1、标准盒模型中 width 指的是内容区域 content 的宽度

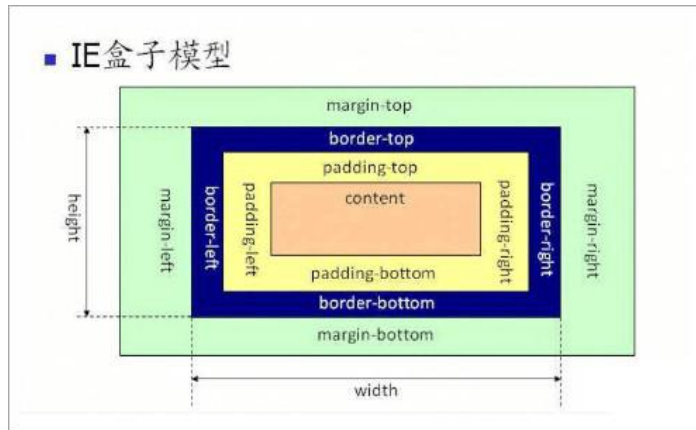
height 指的是内容区域 content 的高度

标准盒模型下盒子的大小 = content + border + padding + margin



2、怪异盒模型中的 width 指的是内容、边框、内边距总的宽度（content + border + padding）； height 指的是内容、边框、内边距总的高度

怪异盒模型下盒子的大小 = width（content + border + padding） + margin



除此之外，我们还可以通过属性 `box-sizing` 来设置盒子模型的解析模式

可以为 `box-sizing` 赋两个值：

`content-box`：默认值，`border` 和 `padding` 不算到 `width` 范围内，可以理解为是

W3c 的标准模型(default)。总宽=`width`+`padding`+`border`+`margin`

`border-box`：`border` 和 `padding` 划归到 `width` 范围内，可以理解为是 IE 的怪异盒

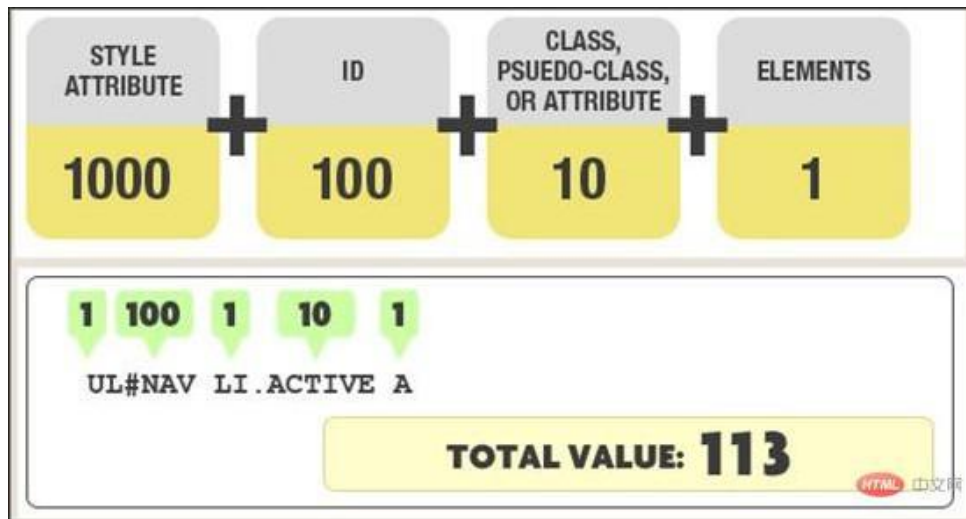
模型，总宽=`width`+`margin`

5、CSS 中选择器的优先级以及 CSS 权重如何计算？（必会）

！ Important > 行内样式 > ID 选择器 > 类选择器 > 标签 > 通配符 > 继承 > 浏览器默认属性

权重

CSS 权重是由四个数值决定，看一张图比较好解释：



图里是英文的，翻译过来分别介绍一下，4个等级的定义如下：

第一等：内联样式，如：style="color:red;"，权值为 1000。（该方法会造成 css 难以管理，所以不推荐使用）

第二等：ID 选择器，如：#header，权值为 0100

第三等：类、伪类、属性选择器如：.bar，权值为 0010

第四等：标签、伪元素选择器，如：div ::first-line 权值为 0001

最后把这些值加起来，再就是当前元素的权重了

其他：

无条件优先的属性只需要在属性后面使用!important。它会覆盖页面内任何位置定义的元素样式。（ie6 支持上有些 bug）

通配符，子选择器，相邻选择器等。如*，>,+，权值为 0000.

继承的样式没有权值

CSS 权重计算方式

计算选择符中的 ID 选择器的数量 (=a)

计算选择符中类、属性和伪类选择器的数量 (=b)

计算选择符中标签和伪元素选择器的数量 (=c)

忽略全局选择器

在分别计算 a、b、c 的值后，按顺序连接 abc 三个数字组成一个新的数字，改值即为所计算的选择符的权重。如果两个选择符的计算权重值相同，则采取“就近原则”。

示例：

```
div#app.child[name="appName"] /a=1,b=2,c=1 —>权重 = 1 + 100 + 10 +10 = 121/
```

6、列举 5 个以上的 H5input 元素 type 属性值？（必会）

值	描述
button	定义可点击的按钮（大多与 JavaScript 使用来启动脚本）
checkbox	定义复选框。
color	定义拾色器。
date	定义日期字段（带有 calendar 控件）
month	定义日期字段的月（带有 calendar 控件）
time	定义日期字段的时、分、秒（带有 time 控件）
email	定义用于 e-mail 地址的文本字段
file	定义输入字段和“浏览...”按钮，供文件上传
hidden	定义隐藏输入字段

image	定义图像作为提交按钮
number	定义带有 spinner 控件的数字字段
password	定义密码字段。字段中的字符会被遮蔽。
radio	定义单选按钮。
search	定义用于搜索的文本字段。
submit	定义提交按钮。提交按钮向服务器发送数据。
text	默认。定义单行输入字段，用户可在其中输入文本。默认是 20 个字符。
url	定义用于 URL 的文本字段。

7、CSS 中哪些属性可继承，哪些不可以？（必会）

能继承的属性

1. 字体系列属性:font、font-family、font-weight、font-size、font-style;
2. 文本系列属性:
 - 2.1) 内联元素: color、line-height、word-spacing、letter-spacing、text-transform;
 - 2.2) 块级元素: text-indent、text-align;
3. 元素可见性: visibility

4. 表格布局属性: caption-side、border-collapse、border-spacing、empty-cells、
table-layout;

5. 列表布局属性: list-style

不能继承的属性

1. display: 规定元素应该生成的框的类型;

2. 文本属性: vertical-align、text-decoration;

3. 盒子模型的属性: width、height、margin 、border、padding;

4. 背景属性: background、background-color、background-image;

5. 定位属性: float、clear、position、top、right、bottom、left、min-width、
min-height、max-width、max-height、overflow、clip;

8、CSS 单位中 px、em 和 rem 的区别? (必会)

1、px 像素 (Pixel) 。绝对单位。像素 px 是相对于显示器屏幕分辨率而言的, 是一个虚拟长度单位, 是计算机系统的数字化图像长度单位

2、em 是相对长度单位, 相对于当前对象内文本的字体尺寸。如当前对行内文本的字体尺寸未被人为设置, 则相对于浏览器的默认字体尺寸。它会继承父级元素的字体大小, 因此并不是一个固定的值

3、rem 是 CSS3 新增的一个相对单位 (root em, 根 em) , 使用 rem 为元素设定字体大小时, 仍然是相对大小, 但相对的只是 HTML 根元素

4、区别:

IE 无法调整那些使用 px 作为单位的字体大小, 而 em 和 rem 可以缩放, rem

相对的只是 HTML 根元素。这个单位可谓集相对大小和绝对大小的优点于一身，通过它既可以做到只修改根元素就成比例地调整所有字体大小，又可以避免字体大小逐层复合的连锁反应。目前，除了 IE8 及更早版本外，所有浏览器均已支持 rem

9、rem 适配方法如何计算 HTML 跟字号及适配方案？（必会）

通用方案

- 1、设置根 font-size: 625% (或其它自定的值，但换算规则 1rem 不能小于 12px)
- 2、通过媒体查询分别设置每个屏幕的根 font-size
- 3、CSS 直接除以 2 再除以 100 即可换算为 rem

优：有一定适用性，换算也较为简单

劣：有兼容性的坑，对不同手机适配不是非常精准；需要设置多个媒体查询来适应不同手机，单某款手机尺寸不在设置范围之内，会导致无法适配

网易方案

- 1、拿到设计稿除以 100，得到宽度 rem 值
- 2、通过给 html 的 style 设置 font-size，把 1 里面得到的宽度 rem 值代入

```
x document.documentElement.style.fontSize =  
document.documentElement.clientWidth / x + 'px';
```

- 3、设计稿 px/100 即可换算为 rem

优：通过动态根 font-size 来做适配，基本无兼容性问题，适配较为精准，换算简便

劣：无 viewport 缩放，且针对 iPhone 的 Retina 屏没有做适配，导致对一些手机的适配不是很到位

手淘方案

1、拿到设计稿除以 10，得到 font-size 基准值

2、引入 flexible

3、不要设置 meta 的 viewport 缩放值

4、设计稿 px/ font-size 基准值，即可换算为 rem

优：通过动态根 font-size、viewpor、dpr 来做适配，无兼容性问题，适配精准。

劣：需要根据设计稿进行基准值换算，在不使用 sublime text 编辑器插件开发时，单位计算复杂

10、Display: none 与 visibility: hidden 的区别？（必会）

最常用的为 display:none 和 visibility:hidden

display:none 设置该属性后，该元素下的元素都会隐藏，占据的空间消失

visibility:hidden 设置该元素后，元素虽然不可见了，但是依然占据空间的位置

区别

1.visibility 具有继承性，其子元素也会继承此属性，若设置 visibility:visible，则子元素会显示

2.visibility 不会影响计数器的计算，虽然隐藏掉了，但是计数器依然继续运行着。

3.在 CSS3 的 transition 中支持 visibility 属性，但是不支持 display，因为 transition 可以延迟执行，因此配合 visibility 使用纯 CSS 实现 hover 延时显示效果可以提高用户体验

户 体验

4.display:none 会引起回流(重排)和重绘 visibility:hidden 会引起重绘

11、Position 的值有哪些，分别有哪些作用？（必会）

static：默认值

不脱离文档流，top, right, bottom, left 等属性不生效

绝对定位：absolute

绝对定位的关键是找对参照物，要成为绝对定位元素的参照物必须满足以下两个条件：

1.参照物和绝对定位元素必须是包含与被包含关系；

2.该参照物必须具有定位属性；

如果找不到满足以上两个条件的父包含块，那么相对于浏览器窗口进行定位

注：设置了 position:absolute;属性后,元素会脱离正常文档流,不在占据空间;左右 margin

为 auto 将会失效；我们通过 left、top、bottom、right 来决定元素位置

相对定位：relative

参照物：元素偏移前位置

注：设置了相对定位，左右 margin 为 auto 仍然有效、并且不会脱离文档流。

固定定位：fixed

参照物：浏览器窗口；

注：固定定位会脱离文档流；

当绝对定位和固定定位参照物都是浏览器窗口时的区别： 当出现滚动条时，固定定位的元

素不会跟随滚动条滚动，绝对定位会跟随滚动条滚动

12、为什么会出现浮动？浮动元素会引起什么问题？如何清除浮动？（必会）

浮动定位将元素排除在普通流之外，即元素脱离文档流，不占据空间。浮动元素碰到包含它的边框或者浮动元素的边框停留

为什么需要清除浮动

- 1、父元素的高度无法被撑开，影响与父元素同级的元素；
- 2、与浮动元素同级的非浮动元素（内联元素）会跟随其后；
- 3、若非第一个元素浮动，则该元素之前的元素也需要浮动，否则会影响页面显示的结构解决方法

清除浮动的方式

- 1、使用 CSS 中的 `clear:both;`（放一个空的 `div`，并设置上述 `css`），属性来清除元素的浮动可解决 2、3 问题
- 2、对于问题 1，添加如下样式，给父元素添加 `clearfix` 样式：`.clearfix:after{content: ".";display: block;height: 0;clear: both;visibility: hidden;}.clearfix{display: inline-block;} /* for IE/Mac */`
- 3、给父级元素设置双伪元素；

```
<div class="container clearfix">

    <div class="wrap">aaa</div>

</div>

.clearfix:after{
```

```
content:"";          /*设置内容为空*/

height:0;            /*高度为 0*/

line-height:0;       /*行高为 0*/

display:block;       /*将文本转为块级元素*/

visibility:hidden;   /*将元素隐藏*/

clear:both;          /*清除浮动*/

}

.clearfix{

    zoom:1;           /*为了兼容 IE*/

}
```

4、给父级元素设置 overflow: hidden; 或 overflow: auto; 本质是构建一个 BFC

13、简述弹性盒子 flex 布局及 rem 布局？（必会）

rem 是 CSS3 新增的一个相对单位，相对于根节点(html)字体大小的值，r 就是 root

html{font-size:10px} 则 2rem=20px

通过它就可以做到只修改根元素的大小，就能成比例地调整所有的字体大小，只依赖 html 字体的大小

适配方案步骤：

1、首先动态计算 html 的 font-size

2、将所有的 px 换算成 rem(计算过程请看下面代码和注释 (注意: rem 的换算是根据

设计图稿的像素计算的, 下面的计算只是动态计算 html 的 font-size 大小) ,

请看下面的注意事项

```
<meta name="viewport" content="width=device-width,user-scalable=no"/>
<style>
  body{
    margin: 0;
  }
  div{
    /*width: 80px;*/
    height: 100px;
    width: 4rem;
    height: 4rem;
    /*1rem=20;  nrem=80;    n=80/rem;    n=80/20;    n=4*/
    background: green;
    float: left;
  }
</style>
<body>
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</body>
<script>
  (function(){
    var html=document.documentElement;
    var width=html.getBoundingClientRect().width; //获取屏幕宽度(设备独立像素), 如
iPhone6 为 414
    html.style.fontSize=width/16+'px'; //html font-size = 20px
    //iphone5 下    1rem=20    这里之所以除以 16, 是因为要把宽度分成 16 份, 这个数并
没有固定, 一般选 15, 16, 以 ipone5 为准是 16, 因为一除可以得到整数 20, 好计算。
  })();
</script>
```

注意:

1、必需动态的去设置 html 的大小, 才能适配

2、根据页面的宽度除以一个系数，把算出的这个值赋给 html 的 font-size 属性，rem 换

算值是根据 psd 设计图的宽度/系数的 rem 系数

以 640px 设计稿和 750px 的视觉稿，网易这样处理的：

```
var width = document.documentElement.clientWidth; // 屏幕的布局视口宽度
var rem = width / 7.5; // 750px 设计稿将布局视口分为 7.5 份
var rem = width / 6.4; // 640px 设计稿将布局视口分为 6.4 份
```

这样不管是 750px 设计稿还是 640px 设计稿，1rem 等于设计稿上的 100px。故 px

转换 rem 时： $1\text{rem} = 1\text{px} * 0.01$;

在 750px 设计稿上：

```
设计稿上 75px 对应 0.75rem, 距离占设计稿的 10%;
在 ipone6 上:
width = document.documentElement.clientWidth = 375px;
1rem = 375px / 7.5 = 50px;
0.75rem = 37.5px; (37.5/375=10%; 占屏幕 10%)
在 ipone5 上:
width = document.documentElement.clientWidth = 320px;
rem = 320px / 7.5 = 42.667px;
0.75rem = 32px; (32/320=10%; 占屏幕 10%)
```

故对于设计稿上任何一个尺寸换成 rem 后，在任何屏下对应的尺寸占屏幕宽度的百分比相

同。故这种布局可以百分比还原设计图

2.1) 为什么要除一个数字，原因是：一个页面里，不可能全都是整屏的元素，肯定有

一行中放多个元素。所以就把一行分成 n 份

2.2) 不除一个数字的话，那 1 个 rem 就是屏幕的宽度，这个值太大，如果一个元素

的宽度比它小的话，就不方便计算

2.3) 这个系数，自己定。多少都可以，但是建议给一个能整除的值（这个能整除的数，

是还要根据设计稿能整除的数。）

3、对于切的图片，尺寸是根据设计图的尺寸宽度的，显示起来会很大，如果是 `img` 标签，

可以设置宽度为切出的图片尺寸，换算成 `rem`，如果是 `background-img`，用

`background-size` 属性，设置设计图尺寸宽高，换算成 `rem` 进行图片的缩放适配。

对于上述的第二点，根据设计稿动态转换 `rem`，这里说一下，前面的计算是动态的设置 `html` 的 `font-size` 的大小，这是根据设备的独立像素计算的。而设计稿往往是根据物理像素，即设备像素设计的，往往很大，是 750px 及以上，所以在转换 `rem` 的时候，转换是根据 `psd` 设计稿的像素进行转换，即 $1\text{rem} = \text{设计稿像素宽度} / \text{系数}$ ，例如，如果是 1080px 的设计稿，那么，就用 $1\text{rem} = 1080 / 18 = 60\text{px}$ （这里用 18 做系数，是因为能整除），然后布局的时候就根据设计稿的元素尺寸转换，例如设计稿一个元素的高为 60px，那么就可以转化为 1rem 了

特点：

- 1、所有有单位的属性会根据屏幕的尺寸自动计算大小
- 2、同样一个元素，在不同的设备下的大小是不一样的。在尺寸小的设备下显示的小，在尺寸大的设备下显示的大
- 3、一般以 `iphone6` 为基准，以它的宽度 750 除上一个系数，再去算 `rem`

Tips：上述步骤 2 中换算可以通过 Hbuilder 将 `px` 自动转 `rem` 以及通过 `less` 自动计算成 `rem`，`sublime` 也可以通过插件进行自动转换

3.1) 打开 Hbuilder，顶部栏的工具》选项》Hbuilder》代码助手》`px` 自动转 `rem` 设置

3.2) `less` 自动转换：Hbuilder 也可以将 `less` 文件自动转成 `css` 文件。`less` 文件的书写如下所示

```
比如想设置宽度为 187px，高度为 100px 的元素，可以通过下面方式计算适配
@rem: 25rem; /*这是 1rem = X px 的 X 的值，但是用了 rem 做单位而已*/
div{
    width: 187/@rem;
    height: 100/@rem;
}
```

弹性布局适配(会配合 rem 适配使用)

兼容情况

IE10 及以上、ios9 及以上、android4.4 及以上版本支持

特点

- 1、默认所有子元素都会在一行中显示，即使给子元素一个很大的宽度
- 2、父级加了这条属性，子级的 float、vertical-align 就会失效
- 3、如果兼容低版本的机型要加前缀-webkit-，包括后面讲的所有属性

容器属性（父元素样式） 具体看菜鸟教程或阮一峰的教程，这里说一下一些重点知识

3.1) flex-direction: 子元素排列方向（主轴的方向，如果设置了 column，则意味着
主轴旋转了 90 度）

3.2) flex-wrap: 换行方式

3.3) flex-flow: 以上两种方式的简写

3.4) justify-content: 水平对齐方式（子元素在主轴上的对齐方式）

3.5) align-items: 垂直对齐方式（子元素在交叉轴上的对齐方式）

3.6) align-content: 多行垂直对齐方式（多根轴线的对齐方式）

项目属性（子元素样式）

- 1、order: 排列位置 //如果有两个的值是相等，按书写顺序排列
- 2、flex-grow: 扩展比例

flex-grow 当父级的宽度大于所有子元素宽度之和时，根据父级的剩余空间，设置子元素的扩展比例（设置后，元素给的固定宽度会被覆盖）它是一个系数默认为 0，即如果存在剩余空间也不扩展

剩余空间：剩余空间=父级的宽度-所有子元素的宽度和

注意：如果没有设置初始宽度，也没有内容，则默认为 0，否则为内容的宽度。例如设置了文字，会撑开有初始宽度

子元素宽度计算公式

子元素的宽度=（父级的宽度-所有子元素的宽度和）/所有子元素的 flex-grow 属性值之和*子元素的 flex-grow 属性值+子元素初始宽度

3、flex-shrink：收缩比例

flex-shrink 当所有子元素宽度之和大于父级宽度的时候，根据超出的空间，设置子元素的收缩比例（设置后，元素给的固定宽度会被覆盖）它是一个系数默认为 1，如果给个 0 的话，就不会收缩

超出空间：超出空间=所有子元素的宽度和-父级的宽度

子元素宽度计算公式

- 1、算出超出空间，所有子元素的宽度和-父级的宽度
- 2、子元素的初始宽度*子元素的 flex-shrink 值
- 3、算出第二步所有结果的和
- 4、每个子元素的第二步/第三步*第一步
- 5、子元素的初始宽度-第四步

flex-basis：元素的大小

flex：以上三个属性的简写

align-self：单独的垂直对齐方式（交叉轴方向上）

14、如何解决 margin “塌陷”？（必会）

外边距塌陷共有两种情况：

第一种情况：两个同级元素，垂直排列，上面的盒子给 margin-bottom 下面的盒子给 margin-top，那么他们两个的间距会重叠，以大的那个计算。解决这种情况

的方法为：两个外边距不同时出现

第二种情况：两个父子元素，内部的盒子给 margin-top，其父级也会受到影响，同时产生上边距，父子元素会进行粘连，决绝这种情况的方法为：父级添加一个 css 属性，overflow：

hidden，禁止超出

外边距重叠就是 margin-collapse

解决方案：

1、为父盒子设置 border，为外层添加 border 后父子盒子就不是真正意义上的贴合（可

以设置成透明：border: 1px solid transparent)；

2、为父盒子添加 overflow: hidden;

3、为父盒子设定 padding 值;

4、为父盒子添加 position: fixed;

5、为父盒子添加 display: table;

6、利用伪元素给父元素的前面添加一个空元素

```
.father::before {  
    content:"";  
    display:table;  
}
```

15、less 和 Scss 的配置使用以及特点？（必会）

less

安装依赖

1 npm install less less-loader --save

或者

2 cnpm install less less-loader --save

修改配置

在 vue 项目中 build/webpack.base.conf.js:

modules 对象的 rules 数组中最后添加

```
{
  test: /\.less$/,
  loader: "style-loader!css-loader!less-loader"
}
```

引入

在每个想要使用 less 的 vue 文件中

style 加上 lang= "less"

1 <style lang="less" scoped>

2 </style>

SCSS

SCSS 即是 SASS 的新语法，是 Sassy CSS 的简写，是 CSS3 语法的超集，也就是说所有有

效的 CSS3 样式也同样适合于 SASS

SASS 是 CSS3 的一个扩展，增加了规则嵌套、变量、混合、选择器继承等等。通过使用命令行的工具或 WEB 框架插件把它转换成标准的、格式良好的 CSS 代码

SCSS 是 SASS 3 引入新的语法，其语法完全兼容 CSS3，并且继承了 SASS 的强大功能。

唯一不同的是，SCSS 需要使用分号和花括号而不是换行和缩进

SCSS 对空白符号不敏感

安装步骤：

```
npm install node-sass --save-dev //安装 node-sass
```

```
npm install sass-loader --save-dev //安装 sass-loader
```

```
npm install style-loader --save-dev //安装 style-loader
```

出现以下问题可能是版本错误

Modele build failed: TypeError: this.getResolve is not a function at Object.loader...

处理方法

将 "sass-loader": "^8.0.0", 更换成了 "sass-loader": "^7.3.1"

package.json 中查找替换

```
npm install
```

```
npm run dev
```

特性：

一、（节点）可嵌套性

这个是基础，用的太多太多了，必须掌握

二、变量

变量以\$开头（通常网站会有基础变量，譬如基础字体，基础色调等，可以将他们赋值给一个变量，以后调用变量就好了，很类似js里的变量）

三、 Mixins(混合@mixin)：可重用性高，可以注入任何东西

注意点：

1、可以相互调用，但是不能拿自己调用自己，形成递归

2、通过@include 引用

四、 @extend：允许一个选择器继承另一个选择器

五、 @function:函数功能，用户使用@function 可以去编写自己的函数（常用）

使用语法： 使用 @function+函数名称，每个函数都需要有返回值的内容

六、引用父元素&：在编译时，&将被替换成父选择符（常用）

七、计算功能（会用 但是不多吧）

八、组合连接： #{} ： 变量连接字符串（目前用到的是这个）

九、循环语句：（很少用到）

十、if 语句：（很少用到）

16、::before 和::after 中双冒号和单冒号有什么区别、作用？（必会）

区别

在 CSS 中伪类一直用 : 表示，如 :hover, :active 等

伪元素在 CSS1 中已存在，当时语法是用 : 表示，如 :before 和 :after

后来在 CSS3 中修订，伪元素用 :: 表示，如 ::before 和 ::after，以此区分伪元素和伪类

由于低版本 IE 对双冒号不兼容，开发者为了兼容性各浏览器，继续使用使用 :after 这种老语法表示伪元素

单冒号 (:) 用于 CSS3 的伪类

双冒号 (::) 用于 CSS3 的伪元素

想让插入的内容出现在其它内容前，使用::before，否则，使用::after；

在代码顺序上，::after 生成的内容也比::before 生成的内容靠后

如果按堆栈视角，::after 生成的内容会在::before 生成的内容之上

作用：

::before 和::after 的主要作用是在元素内容前后加上指定内容

伪类与伪元素都是用于向选择器加特殊效果

伪类与伪元素的本质区别就是是否抽象创造了新元素

伪类只要不是互斥可以叠加使用

伪元素在一个选择器中只能出现一次，并且只能出现在末尾

伪类与伪元素优先级分别与类、标签优先级相同

17、CSS3 新增伪类，以及伪元素？（必会）

CSS3 新增伪类

p:first-of-type 选择属于其父元素的首个<p>元素的每个<p>元素

p:last-of-type 选择属于其父元素的最后<p>元素的每个<p>元素

p:nth-child(n) 选择属于其父元素的第 n 个子元素的每个<p>元素

p:nth-last-child(n) 选择属于其父元素的倒数第 n 个子元素的每个<p>元素

p:nth-of-type(n) 选择属于其父元素第 n 个<p>元素的每个<p>元素

p:nth-last-of-type(n) 选择属于其父元素倒数第 n 个<p>元素的每个<p>元素

p:last-child 选择属于其父元素最后一个子元素的每个<p>元素

p:target 选择当前活动的<p>元素

:not(p) 选择非<p>元素的每个元素

:enabled 控制表单控件的可用状态

:disabled 控制表单控件的禁用状态

:checked 单选框或复选框被选中

伪元素

::first-letter 将样式添加到文本的首字母

::first-line 将样式添加到文本的首行

::before 在某元素之前插入某些内容

::after 在某元素之后插入某些内容

18、Bootstrap 栅格系统的工作原理？（必会）

原理

- 1、行（row）必须包含在.container(固定宽度)或.container-fluid(100%宽度)中，以便为其赋予合适的排列（alignment）和内补（padding）
- 2、通过行（row）在水平方向创建一组列（column）
- 3、自己内容应当放置于列（column）内，并且，只有列可以作为行（row）的直接子元素

-
- 4、类似.row 和.col-xs-4 这种预定义的类，可以用来快速创建栅格布局。Bootstrap 源码中定义的 mixin 也可以用来创建语义化布局
 - 5、通过为列设置 padding 属性，从而创建列与列之间的间隔（gutter）。通过为.row 元素设置负值 margin 从而抵消为.container 元素设置的 padding,也就间接为行(row)所包含的列（column）抵消掉了 padding
 - 6、栅格系统的列是通过指定 1 到 12 的值来表示其跨越范围。例如三个等宽的列可以使用三个.col-xs-4 来创建
 - 7、如果一行（row）中包含的列（column）大于 12，多余的列所在的元素将作为一个整体另起一行排列
 - 8、栅格类适用于与屏幕宽度大于或等于分界点大小的设备，并且针对小屏幕覆盖栅格类

使用 Bootstrap 响应式布局

首先需要在 head 中引入 meta 标签，添加 viewport 属性，content 中宽度等于设备宽度，initial-scale:页面首次被显示可见区域的缩放级别，取值 1 则页面按实际尺寸显示，无任何缩放；maximum-scale:允许用户缩放到的最小比例；user-scalable:用户是否可以手动缩放。代码如下：

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=no">
<link rel="stylesheet" type="text/css" href="/stylesheets/bootstrap.min.css">
```

下面为使用 Bootstrap 布局的页面（登录表单界面），针对的是手机超小屏幕（iphone5s）和 PC 屏幕（>=1200px）。col-xs-12:小屏幕占 12 列大小，col-lg-5: 大屏幕占 5 列大小，col-lg-offset-3:大屏幕缩进 3 列大小。这是一个比较简单的实例，想要适应其他屏幕如平

板可添加 col-md-* 属性，大屏手机可添加 col-sm-*属性。具体的屏幕使用哪个属性，可参考上面图上的针对不同屏幕 Bootstrap 栅格系统的不同使用。

```
<div class="container-fluid login">
  <div class="row">
    <div class="col-xs-12 col-sm-12 col-md-8 col-lg-5 col-lg-offset-3">
      <form class="form-horizontal loginForm">
        <h3 class="form-signin-heading">用户登录</h3>
        <div class="form-group">
          <label for="email" class="col-sm-2 col-xs-3 control-label">邮箱</label>
          <div class="col-sm-8 col-xs-8">
            <input type="text" class="form-control" name="email" placeholder="请输入邮箱">
            <span class="glyphicon glyphicon-ok form-control-feedback"
aria-hidden="true"></span>
          </div>
        </div>
        <div class="form-group">
          <label for="password" class="col-sm-2 col-xs-3 control-label">密码</label>
          <div class="col-sm-8 col-xs-8">
            <input type="password" class="form-control" name="password"
placeholder="请输入密码">
            <span class="glyphicon glyphicon-ok form-control-feedback"
aria-hidden="true"></span>
          </div>
        </div>
        <div class="form-group">
          <div class="col-sm-offset-2 col-sm-4 col-xs-4">
            <div class="checkbox">
              <label>
                <input type="checkbox">记住我 </label>
              </div>
            </div>
            <div class="col-sm-4 col-xs-4 control-label">
              <a href="resetPwd.html" id="forget">忘记密码? </a>
            </div>
          </div>
          <div class="form-group">
            <div class="col-sm-12 col-lg-12">
              <button type="button" class="btn btn-primary btn-block" id="submit">
```

```
登录</button>
    </div>
</div>
</form>
</div>
</div>
```

19、自己手动封装响应式布局方案？（必会）

方案

首先，在网页代码的头部，加入一行 viewport 元标签。

```
<meta name="viewport"
content="width=device-width,initial-scale=1.0,maximum-scale=1.0,minimum-scale=1.0,user-scalable=no" />
```

viewport 是网页默认的宽度和高度，上面这行代码的意思是，网页宽度默认等于屏幕宽度

(width=device-width)，原始缩放比例 (initial-scale=1) 为 1.0，即网页初始大小占屏幕面积的 100%

所有主流浏览器都支持这个设置，包括 IE9。对于那些老式浏览器（主要是 IE6、7、8），

需要使用 css3-mediaqueries.js

```
<![if lt IE 9]><script src="http://css3-mediaqueries-js.googlecode.com/svn/trunk/css3-mediaqueries.js"></script><![endif]>
```

不使用绝对宽度

由于网页会根据屏幕宽度调整布局，所以不能使用绝对宽度的布局，也不能使用具有绝对宽度的元素。这一条非常重要

具体说，CSS 代码不能指定像素宽度：width:xxx px;只能指定百分比宽度：width: xx%;

或者 width:auto;

相对大小的字体

字体也不能使用绝对大小 (px) , 而只能使用相对大小 (em)

```
body {font: normal 100% Helvetica, Arial, sans-serif;}
```

上面的代码指定, 字体大小是页面默认大小的 100%, 即 16 像素。h1 {font-size: 1.5em;}

然后, h1 的大小是默认大小的 1.5 倍, 即 24 像素 (24/16=1.5)。small {font-size: 0.875em;}

small 元素的大小是默认大小的 0.875 倍, 即 14 像素 (14/16=0.875)

流动布局 (fluid grid)

“流动布局” 的含义是, 各个区块的位置都是浮动的, 不是固定不变的。

```
.main{float: right;width: 70%;}
```

```
.leftBar{float: left;width: 25%;}
```

float 的好处是, 如果宽度太小, 放不下两个元素, 后面的元素会自动滚动到前面元素的下方, 不会在水平方向 overflow (溢出) , 避免了水平滚动条的出现。

另外, 绝对定位 (position: absolute) 的使用, 也要非常小心

选择加载 CSS

“自适应网页设计” 的核心, 就是 CSS3 引入的 Media Query 模块

它的意思就是, 自动探测屏幕宽度, 然后加载相应的 CSS 文件

```
<link rel="stylesheet" type="text/css" media="screen and (max-device-width: 400px)"
href="tinyScreen.css"/>
```

上面的代码意思是, 如果屏幕宽度小于 400 像素 (max-device-width: 400px) , 就加载 tinyScreen.css 文件

```
<link rel="stylesheet" type="text/css" media="screen and (min-width: 400px) and
(max-device-width: 600px)" href="smallScreen.css"/>
```

如果屏幕宽度在 400 像素到 600 像素之间, 则加载 smallScreen.css 文件

除了用 html 标签加载 CSS 文件, 还可以在现有 CSS 文件中加载

```
@import url("tinyScreen.css") screen and (max-device-width: 400px);
```

CSS 的 @media 规则

同一个 CSS 文件中，也可以根据不同的屏幕分辨率，选择应用不同的 CSS 规则

```
@media screen and (max-device-width: 400px) { .column {float: none;width: auto;} #sidebar { display: none'}}
```

上面的代码意思是，如果屏幕宽度小于 400 像素，则 column 块取消浮动 (float:none) 、

宽度自动调节 (width:auto) ， sidebar 块不显示 (display:none)

图片的自适应 (fluid image)

除了布局和文本，“自适应网页设计”还必须实现图片的自动缩放

这只要一行 CSS 代码：img { max-width: 100%;}

这行代码对于大多数嵌入网页的视频也有效，所以可以写成：

```
img, object { max-width: 100%;}
```

老版本的 IE 不支持 max-width，所以只好写成：img { width: 100%; }

此外，windows 平台缩放图片时，可能出现图像失真现象。这时，可以尝试使用 IE 的专有

命令：

```
img { -ms-interpolation-mode: bicubic; }
```

或者，Ethan Marcotte 的 imgSizer.js

```
addLoadEvent(function() { var imgs = document.getElementById("content").getElementsByTagName("img"); imgSizer.collate(imgs); });
```

不过，有条件的话，最好还是根据不同大小的屏幕，加载不同分辨率的图片

20、BFC 是什么？（高薪常问）

定义

BFC(Block formatting context)直译为"块级格式化上下文"。它是一个独立的渲染区域，只有 Block-level box 参与，它规定了内部的 Block-level Box 如何布局，并且与这个区域外部毫不相干

布局规则

- 1、内部的 Box 会在垂直方向，一个接一个地放置
- 2、Box 垂直方向的距离由 margin 决定。属于同一个 BFC 的两个相邻 Box 的 margin 会发生重叠
- 3、每个元素的 margin box 的左边，与包含块 border box 的左边相接触(对于从左往右的格式化，否则相反)。即使存在浮动也是如此
- 4、BFC 的区域不会与 float box 重叠
- 5、BFC 就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面的元素。反之也如此
- 6、计算 BFC 的高度时，浮动元素也参与计算

哪些元素会生成 BFC：

- 1、根元素
- 2、float 属性不为 none
- 3、position 为 absolute 或 fixed
- 4、display 为 inline-block, table-cell, table-caption, flex, inline-flex
- 5、overflow 不为 visible

21、什么是渐进增强和优雅降级?它们有什么不同? (了解)

优雅降级和渐进增强印象中是随着 CSS3 流出来的一个概念。由于低级浏览器不支持 CSS3, 但 CSS3 的效果又太优秀不忍放弃, 所以在高级浏览中使用 CSS3 而低级浏览器只保证最基本的功能。关键的区别 是他们所侧重的内容, 以及这种不同造成的工作流程的差异

举个例子:

```
a{

display: block;

width: 200px;

height: 100px;

background: aquamarine;

/*我就是要用这个新 css 属性*/

transition: all 1s ease 0s;

/*可是发现了一些低版本浏览器不支持怎么吧*/

/*往下兼容*/

-webkit-transition: all 1s ease 0s;

-moz-transition: all 1s ease 0s;

-o-transition: all 1s ease 0s;

/*那么通常这样考虑的和这样的侧重点出发的 css 就是优雅降级*/

}

a: hover{
```

```
height: 200px;

}

/*那如果我们的产品要求我们要重低版本的浏览器兼容开始*/

a{

/*优先考虑低版本的*/

-webkit-transition: all 1s ease 0s;

-moz-transition: all 1s ease 0s;

-o-transition: all 1s ease 0s;

/*高版本的就肯定是渐进渐强*/

transition: all 1s ease 0s;

}
```

“优雅降级” 观点认为应该针对那些最高级、最完善的浏览器来设计网站

“渐进增强” 观点则认为应关注于内容本身

22、iframe 有哪些优缺点？（了解）

iframe 的优点：

- 1、iframe 能够原封不动的把嵌入的网页展现出来
- 2、如果有多个网页引用 iframe，那么只需要修改 iframe 的内容，就可以实现调用每一个页面的更改，方便快捷
- 3、网页如果为了统一风格，头部和版本都是一样的，就可以写成一个页面，用 iframe 嵌套，可以增加代码的可重用

-
- 4、如果遇到加载缓慢的第三方内容，如图标或广告，这些问题可以由 iframe 来解决。
 - 5、iframe 会堵塞主页面的 Onload 事件
 - 6、iframe 和主页面共享连接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载

iframe 的缺点：

- 1、iframe 会阻塞主页面的 Onload 事件；
- 2、iframe 和主页面共享链接池，而浏览器对相同域的连接有限制，所以会影响页面的并行加载；
- 3、使用 iframe 之前需要考虑这两个缺点，如果需要使用 iframe，最好是通过 JavaScript；
- 4、动态给 iframe 添加 src 属性值，这样可以可以绕开以上两个问题
- 5、不利于 seo
- 6、代码复杂，无法一下被搜索引擎索引到
- 7、iframe 框架页面会增加服务器的 http 请求，对于大型网站不可取。
- 8、很多的移动设备无法完全显示框架，设备兼容性差

23、使用 CSS 怎么让 Chrome 支持小于 12px 的文字比如 10px？ (了解)

作法

针对谷歌浏览器内核，加 webkit 前缀，用 transform:scale()这个属性进行缩放！

```
<style>
  p span{font-size:10px;-webkit-transform:scale(0.8);display:block;}
</style>
<p>
```

```
<span>豪豪豪 10px</span>
</p>
```

JavaScript 基础

1、JavaScript 的基本类型有哪些？引用类型有哪些？null 和 undefined 的区别？（必会）

数据类型

基本数据类型：number、string、boolean、null、undefined

引用数据类型：function、object、Array

区别

undefined:表示变量声明但未初始化时的值

null：表示准备用来保存对象，还没有真正保存对象的值。从逻辑角度看，null 值表示一个空对象指针

JavaScript(ECMAScript 标准)里共有 5 种基本类型:Undefined, Null, Boolean, Number, String, 和一种复杂类型 Object。可以看到 null 和 undefined 分属不同的类型，未初始化的值用 typeof 检测出来是"undefined"(字符串)，而 null 值用 typeof 检测出来是"object"（字符串）。任何时候都不建议显式的设置一个变量为 undefined，但是如果保存对象的变量还没有真正保存对象，应该设置成 null。实际上，undefined 值是派生自 null 值的，ECMAScript 标准规定对二者进行相等性测试要返回 true

2、如何判断 JavaScript 的数据类型？（必会）

JavaScript 数据类型一共有 7 种：

Undefined、Null、Boolean、String、Symbol、Number、Object

除了 Object 之外的 6 种属于原始数据类型。有时，我们还会细分 Object 的类型，比如 Array，

Function，Date，RegExp 等

判断方法

typeof

typeof 可以用来区分除了 Null 类型以外的原始数据类型，对象类型的可以从普通对象里面

识别出函数：

```
typeof undefined // "undefined"
typeof null // "object"
typeof 1 // "number"

typeof "1" // "string"
typeof Symbol() // "symbol"
typeof function() {} // "function"
typeof {} // "object"
```

问题一：typeof 不能识别 null，如何识别 null？

答案：如果想要判断是否为 null，可以直接使用===全等运算符来判断（或者使用下面的

Object.prototype.toString 方法）：

```
let a = null

a === null // true
```

问题二：typeof 作用于未定义的变量，会报错吗？

答案：不会报错，返回"undefined"。

```
typeof randomVariable // "undefined"
```

问题三：typeof Number(1)的返回值是什么？

答案："number"。注意 Number 和 String 作为普通函数调用的时候，是把参数转化为相应的原始数据类型，也就是类似于做一个强制类型转换的操作，而不是默认当做构造函数调用。注意和 Array 区分，Array(...)等价于 new Array(...)

```
typeof Number(1) // "number"
```

```
typeof String("1") // "string"
```

```
Array(1, 2, 3)
```

```
// 等价于
```

```
new Array(1, 2, 3)
```

问题四：typeof new Number(1)的返回值是什么？

答案："object"。

```
typeof new Number(1) // "object"
```

```
typeof new String(1) // "object"
```

instanceof

instanceof 不能用于判断原始数据类型的数据：

```
3 instanceof Number // false
```

```
'3' instanceof String // false
```

```
true instanceof Boolean // false
```

instanceof 可以用来判断对象的类型：

```
var date = new Date()
```

```
date instanceof Date // true

var number = new Number()

number instanceof Number // true

var string = new String()

string instanceof String // true
```

需要注意的是, instanceof 的结果并不一定是可靠的, 因为在 ECMAScript7 规范中可以通过自定义 Symbol.hasInstance 方法来覆盖默认行为。

Object.prototype.toString

```
Object.prototype.toString.call(undefined).slice(8, -1) // "Undefined"

Object.prototype.toString.call(null).slice(8, -1) // "Null"

Object.prototype.toString.call(3).slice(8, -1) // "Number"

Object.prototype.toString.call(new Number(3)).slice(8, -1) // "Number"

Object.prototype.toString.call(true).slice(8, -1) // "Boolean"

Object.prototype.toString.call('3').slice(8, -1) // "String"

Object.prototype.toString.call(Symbol()).slice(8, -1) // "Symbol"
```

由上面的示例可知, 该方法没有办法区分数字类型和数字对象类型, 同理还有字符串类型和字符串对象类型、布尔类型和布尔对象类型

另外, ECMAScript7 规范定义了符号 Symbol.toStringTag, 你可以通过这个符号自定义

Object.prototype.toString 方法的行为:

```
'use strict'

var number = new Number(3)
```

```
number[Symbol.toStringTag] = 'Custom'

Object.prototype.toString.call(number).slice(8, -1) // "Custom"

function a () {}

a[Symbol.toStringTag] = 'Custom'

Object.prototype.toString.call(a).slice(8, -1) // "Custom"

var array = []

array[Symbol.toStringTag] = 'Custom'

Object.prototype.toString.call(array).slice(8, -1) // "Custom"
```

因为 `Object.prototype.toString` 方法可以通过 `Symbol.toStringTag` 属性来覆盖默认行为，所以使用这个方法来判断数据类型也不一定是可靠的

`Array.isArray`

`Array.isArray(value)`可以用来判断 `value` 是否是数组：

```
Array.isArray([]) // true

Array.isArray({}) // false

(function () {console.log(Array.isArray(arguments))})(); // false
```

3、简述创建函数的几种方式？（必会）

第一种（函数声明）：

```
function sum1(num1,num2){
  return num1+num2;
}
```

第二种（函数表达式）

```
var sum2 = function(num1,num2){
```

```
return num1+num2;
}
```

第三种（函数对象方式）

```
var sum3 = new Function("num1","num2","return num1+num2");
```

4、Javascript 创建对象的几种方式？（必会）

1、简单对象的创建 使用对象字面量的方式{}

创建一个对象（最简单，好理解，推荐使用）

代码如下

```
var Cat = {}; //JSON
Cat.name="kity"; //添加属性并赋值
Cat.age=2;
Cat.sayHello=function(){
    alert("hello "+Cat.name+",今年"+Cat["age"]+"岁了");//可以使用 "." 的方式访问属性,
    也可以使用 HashMap 的方式访问
}
Cat.sayHello();//调用对象的（方法）函数
```

2、用 function(函数)来模拟 class

2.1) 创建一个对象，相当于 new 一个类的实例(无参构造函数)

代码如下

```
function Person(){
}
var personOne=new Person();//定义一个 function，如果有 new 关键字去"实例化",那么该 function 可以看作是一个类
personOne.name="dylan";
personOne.hobby="coding";
personOne.work=function(){
    alert(personOne.name+" is coding now...");
}
personOne.work();
```

2.2) 可以使用有参构造函数来实现，这样定义更方便，扩展性更强（推荐使用）

代码如下

```
function Pet(name,age,hobby){
  this.name=name;//this 作用域：当前对象
  this.age=age;
  this.hobby=hobby;
  this.eat=function(){
    alert("我叫"+this.name+"，我喜欢"+this.hobby+"，也是个吃货");
  }
}

var maidou =new Pet("麦兜",5,"睡觉");//实例化/创建对象
maidou.eat();//调用 eat 方法(函数)
```

3、使用工厂方式来创建 (Object 关键字)

代码如下：

```
var wcDog =new Object();
wcDog.name="旺财";
wcDog.age=3;
wcDog.work=function(){
  alert("我是"+wcDog.name+"，汪汪汪.....");
}

wcDog.work();
```

4、使用原型对象的方式 prototype 关键字

代码如下：

```
function Dog(){
}

Dog.prototype.name="旺财";
Dog.prototype.eat=function(){
  alert(this.name+"是个吃货");
}

var wangcai =new Dog();
wangcai.eat();
```

5、混合模式(原型和构造函数)

代码如下：

```
function Car(name,price){
  this.name=name;
  this.price=price;
```

```
}  
Car.prototype.sell=function(){  
alert("我是"+this.name+"，我现在卖"+this.price+"万元");  
}  
var camry =new Car("凯美瑞",27);  
camry.sell();
```

6、动态原型的方式(可以看作是混合模式的一种特例)

代码如下：

```
function Car(name,price){  
this.name=name;  
this.price=price;  
if(typeof Car.sell=="undefined"){  
Car.prototype.sell=function(){  
alert("我是"+this.name+"，我现在卖"+this.price+"万元");  
}  
Car.sell=true;  
}  
}  
var camry =new Car("凯美瑞",27);  
camry.sell();
```

以上几种，是 javascript 中最常用的创建对象的方式

5、请指出 JavaScript 宿主对象和原生对象的区别？（必会）

原生对象

ECMA-262 把本地对象（native object）定义为“独立于宿主环境的 ECMAScript 实现提供的对象”

“本地对象”包含哪些内容：Object、Function、Array、String、Boolean、Number、Date、RegExp、Error、EvalError、RangeError、ReferenceError、SyntaxError、TypeError、URIError

由此可以看出，本地对象就是 ECMA-262 定义的类（引用类型）

内置对象

ECMA-262 把内置对象 (built-in object) 定义为“由 ECMAScript 实现提供的、独立于宿主环境的所有对象，在 ECMAScript 程序开始执行时出现”。这意味着开发者不必明确实例化内置对象，它已被实例化了

同样是“独立于宿主环境”。根据定义我们似乎很难分清“内置对象”与“本地对象”的区别。而 ECMA-262 只定义了两个内置对象，即 Global 和 Math（它们也是本地对象，根据定义，每个内置对象都是本地对象）。如此就可以理解了。内置对象是本地对象的一种

宿主对象

何为“宿主对象”？主要在这个“宿主”的概念上，ECMAScript 中的“宿主”当然就是我们网页的运行环境，即“操作系统”和“浏览器”

所有非本地对象都是宿主对象 (host object)，即由 ECMAScript 实现的宿主环境提供的对象。所有的 BOM 和 DOM 都是宿主对象。因为其对于不同的“宿主”环境所展示的内容不同。其实说白了就是，ECMAScript 官方未定义的对象都属于宿主对象，因为其未定义的对象大多数是自己通过 ECMAScript 程序创建的对象

6、JavaScript 内置的常用对象有哪些？并列举该对象常用的方法？（必会）

对象及方法

Arguments 函数参数集合

Arguments[] 函数参数的数组

Arguments 一个函数的参数和其他属性

Arguments.callee 当前正在运行的函数

Arguments.length 传递给函数的参数的个数

Array 数组

length 属性 动态获取数组长度

join() 将一个数组转成字符串。返回一个字符串。

reverse() 将数组中各元素颠倒顺序

delete 运算符 只能删除数组元素的值，而所占空间还在，总长度没变(arr.length)。

shift() 删除数组中第一个元素，返回删除的那个值，并将长度减 1。

pop() 删除数组中最后一个元素，返回删除的那个值，并将长度减 1。

unshift() 往数组前面添加一个或多个数组元素，长度要改变。arrObj.unshift("a" ,
"b, "c")

push() 往数组结尾添加一个或多个数组元素，长度要改变。arrObj.push("a" , "b" ,
"c")

concat() 连接数组

slice() 返回数组的一部分

sort() 对数组元素进行排序

splice() 插入、删除或替换数组的元素

toLocaleString() 把数组转换成局部字符串

toString() 将数组转换成一个字符串

forEach 遍历所有元素

```
var arr = [1, 2, 3];
```

```
arr.forEach(function(item, index) {  
  
    // 遍历数组的所有元素  
  
    console.log(index, item);  
  
});
```

every 判断所有元素是否都符合条件

```
var arr = [1, 2, 3];  
var arr1 = arr.every(function(item, index) {  
    if (item < 4) {  
        return true;  
    }  
})  
console.log(arr1); // true
```

sort 排序

```
var arr = [1, 5, 2, 7, 3, 4];  
var arr2 = arr.sort(function(a, b) {  
    // 从小到大  
    return a-b;  
    // 从大到小  
    return b-a;  
})  
console.log(arr2); // 1,2,3,4,5,7
```

map 对元素重新组装，生成新数组

```
var arr = [1, 5, 2, 7, 3, 4];  
  
var arr2 = arr.map(function(item, index) {  
  
    return '<b>' + item + '</br>';  
  
})  
  
console.log(arr2);
```

filter 过滤符合条件的元素

```
var arr = [1, 2, 3, 4];
```

```
var arr2 = arr.filter(function(item, index) {
```

```
    if (item>2) {
```

```
        return true;
```

```
    }
```

```
})
```

```
console.log(arr2); // [3, 4]
```

String 字符串对象

Length 获取字符串的长度。如：var len = strObj.length

toLowerCase() 将字符串中的字母转成全小写。如：strObj.toLowerCase()

toUpperCase() 将字符串中的字母转成全大写。如：strObj.toUpperCase()

charAt(index) 返回指定下标位置的一个字符。如果没有找到，则返回空字符串

substr() 在原始字符串，返回一个子字符串

substring() 在原始字符串，返回一个子字符串

区别：''

```
"abcdefgh" .substring(2, 3) = "c"
```

```
"abcdefgh" .substr(2, 3) = "cde"
```

split() 将一个字符串转成数组

charCodeAt() 返回字符串中的第 n 个字符的代码

concat() 连接字符串

`fromCharCode()` 从字符编码创建一个字符串

`indexOf()` 返回一个子字符串在原始字符串中的索引值(查找顺序从左往右查找)。如果没有找到, 则返回-1

`lastIndexOf()` 从后向前检索一个字符串

`localeCompare()` 用本地特定的顺序来比较两个字符串

`match()` 找到一个或多个正则表达式的匹配

`replace()` 替换一个与正则表达式匹配的子串

`search()` 检索与正则表达式相匹配的子串

`slice()` 抽取一个子串

`toLocaleLowerCase()` 把字符串转换小写

`toLocaleUpperCase()` 将字符串转换成大写

`toLowerCase()` 将字符串转换成小写

`toString()` 返回字符串

`toUpperCase()` 将字符串转换成大写

`valueOf()`

Boolean 布尔对象

`Boolean.toString()` 将布尔值转换成字符串

`Boolean.valueOf()` Boolean 对象的布尔值

Date 日期时间

创建 Date 对象的方法

(1) 创建当前(现在)日期对象的实例, 不带任何参数

```
var today = new Date();
```

(2) 创建指定时间戳的日期对象实例，参数是时间戳。

时间戳: 是指某一个时间距离 1970 年 1 月 1 日 0 时 0 分 0 秒, 过去了多少毫秒值(1 秒

=1000 毫秒)

```
var timer = new Date(10000); //时间是 1970 年 1 月 1 日 0 时 0 分 10 秒
```

(3) 指定一个字符串的日期时间信息，参数是一个日期时间字符串

```
var timer = new Date( "2015/5/25 10: 00: 00" );
```

(4) 指定多个数值参数

```
var timer = new Date(2015+100, 4, 25, 10, 20, 0); //顺序为: 年、月、日、
```

时、分、秒, 年、月、日是必须的

方法:

Date.getDate() 返回一个月中的某一天

Date.getDay() 返回一周中的某一天

Date.getFullYear() 返回 Date 对象的年份字段

Date.getHours() 返回 Date 对象的小时字段

Date.getMilliseconds() 返回 Date 对象的毫秒字段

Date.getMinutes() 返回 Date 对象的分钟字段

Date.getMonth() 返回 Date 对象的月份字段

Date.getSeconds() 返回 Date 对象的秒字段

Date.getTime() 返回 Date 对象的毫秒表示

Error 异常对象

Error.message 可以读取的错误消息

Error.name 错误的类型

Error.toString() 把 Error 对象转换成字符串

EvalError 在不正确使用 eval()时抛出

SyntaxError 抛出该错误用来通知语法错误

RangeError 在数字超出合法范围时抛出

ReferenceError 在读取不存在的变量时抛出

TypeError 当一个值的类型错误时，抛出该异常

URIError 由 URI 的编码和解码方法抛出

Function 函数构造器

Function.apply() 将函数作为一个对象的方法调用

Function.arguments[] 传递给函数的参数

Function.call() 将函数作为对象的方法调用

Function.caller 调用当前函数的函数

Function.length 已声明的参数的个数

Function.prototype 对象类的原型

Function.toString() 把函数转换成字符串

Math 数学对象

Math 对象是一个静态对象

Math.PI 圆周率

Math.abs() 绝对值

Math.ceil() 向上取整(整数加 1, 小数去掉)

Math.floor() 向下取整(直接去掉小数)

Math.round() 四舍五入

Math.pow(x, y) 求 x 的 y 次方

Math.sqrt() 求平方根

Number 数值对象

Number.MAX_VALUE 最大数值

Number.MIN_VALUE 最小数值

Number.NaN 特殊的非数字值

Number.NEGATIVE_INFINITY 负无穷大

Number.POSITIVE_INFINITY 正无穷大

Number.toExponential() 用指数计数法格式化数字

Number.toFixed() 采用定点计数法格式化数字

Number.toLocaleString() 把数字转换成本地格式的字符串

Number.toPrecision() 格式化数字的有效位

Number.toString() 将一个数字转换成字符串

Number.valueOf() 返回原始数值

Object 基础对象

Object 含有所有 JavaScript 对象的特性的超类

Object.constructor 对象的构造函数

`Object.hasOwnProperty()` 检查属性是否被继承

`Object.isPrototypeOf()` 一个对象是否是另一个对象的原型

`Object.propertyIsEnumerable()` 是否可以通过 `for/in` 循环看到属性

`Object.toLocaleString()` 返回对象的本地字符串表示

`Object.toString()` 定义一个对象的字符串表示

`Object.valueOf()` 指定对象的原始值

RegExp 正则表达式对象

`RegExp.exec()` 通用的匹配模式

`RegExp.global` 正则表达式是否全局匹配

`RegExp.ignoreCase` 正则表达式是否区分大小写

`RegExp.lastIndex` 下次匹配的起始位置

`RegExp.source` 正则表达式的文本

`RegExp.test()` 检测一个字符串是否匹配某个模式

`RegExp.toString()` 把正则表达式转换成字符串

7、=== 和 ==的区别? (必会)

区别

`===`: 三个等号我们称为等同符, 当等号两边的值为相同类型的时候, 直接比较等号两边的值, 值相同则返回 `true`, 若等号两边的值类型不同时直接返回 `false`。也就是说三个等号既要判断值也要判断类型是否相等

==: 两个等号我们称为等值符，当等号两边的值为相同类型时比较值是否相同，类型不同时会发生类型的自动转换，转换为相同的类型后再作比较。也就是说两个等号只要值相等就可以

8、null, undefined 的区别（必会）

区别

null 表示一个对象被定义了，值为“空值”；

undefined 表示不存在这个值

```
typeof undefined //"undefined"
```

undefined :是一个表示"无"的原始值或者说表示"缺少值"，就是此处应该有一个值，但还没有定义。当尝试读取时会返回 undefined；

例如变量被声明了，但没有赋值时，就等于 undefined

```
typeof null //"object"
```

null：是一个对象(空对象，没有任何属性和方法)；

例如作为函数的参数，表示该函数的参数不是对象；

注意：

在验证 null 时，一定要使用===，因为 == 无法分别 null 和 undefined

undefined 表示"缺少值"，就是此处应该有一个值，但是还没有定义。典型用法是：

- 1、变量被声明了，但没有赋值时，就等于 undefined
- 2、调用函数时，应该提供的参数没有提供，该参数等于 undefined
- 3、对象没有赋值的属性，该属性的值为 undefined

4、函数没有返回值时，默认返回 undefined

null 表示"没有对象"，即该处不应该有值。典型用法是：

4.1) 作为函数的参数，表示该函数的参数不是对象

4.2) 作为对象原型链的终点

9、JavaScript 中什么情况下会返回 undefined 值？（必会）

1、访问声明，但是没有初始化的变量

```
var aaa;  
console.log(aaa); // undefined
```

2、访问不存在的属性

```
var aaa = {};  
console.log(aaa.c);
```

3、访问函数的参数没有被显式的传递值

```
(function (b){  
    console.log(b); // undefined  
})();
```

4、访问任何被设置为 undefined 值的变量

```
var aaa = undefined; console.log(aaa); // undefined
```

5、没有定义 return 的函数隐式返回

```
function aaa(){ console.log(aaa()); // undefined
```

6、函数 return 没有显式的返回任何内容

```
function aaa(){  
    return;  
} console.log(aaa()); // undefined
```

10、如何区分数组和对象？（必会）

方法一：通过 ES6 中的 Array.isArray 来识别

```
Array.isArray([]) //true
Array.isArray({}) //false
```

方法二：通过 instanceof 来识别

```
[] instanceof Array //true
{} instanceof Array //false
```

方法三：通过调用 constructor 来识别

```
{}.constructor //返回 object
[].constructor //返回 Array
```

方法四：通过 Object.prototype.toString.call 方法来识别

```
Object.prototype.toString.call([]) //["object Array"]
Object.prototype.toString.call({}) //["object Object"]
```

11、多维数组降维的几种方法（必会）

(1) 数组字符串化

```
let arr = [[222, 333, 444], [55, 66, 77]]
arr += '';
arr = arr.split(',');

console.log(arr); // ["222", "333", "444", "55", "66", "77"]
```

(2) 递归

```
function reduceDimension(arr){
  let ret = [];
  let toArr = function(arr){
    arr.forEach(function(item){
      item instanceof Array ? toArr(item) : ret.push(item);
    });
  }
  toArr(arr);
  return ret;
}

3、Array.prototype.flat()

var arr1 = [1, 2, [3, 4]];
```

```
arr1.flat();  
// [1, 2, 3, 4]  
  
var arr2 = [1, 2, [3, 4, [5, 6]]];  
arr2.flat();  
// [1, 2, 3, 4, [5, 6]]  
  
var arr3 = [1, 2, [3, 4, [5, 6]]];  
arr3.flat(2);  
// [1, 2, 3, 4, 5, 6]  
  
//使用 Infinity 作为深度，展开任意深度的嵌套数组  
arr3.flat(Infinity);  
// [1, 2, 3, 4, 5, 6]
```

4、使用 stack 无限反嵌套多层嵌套数组

```
var arr1 = [1,2,3,[1,2,3,4, [2,3,4]]];  
function flatten(input) {  
  const stack = [...input];  
  const res = [];  
  while (stack.length) {  
    // 使用 pop 从 stack 中取出并移除值  
    const next = stack.pop();  
    if (Array.isArray(next)) {  
      // 使用 push 送回内层数组中的元素，不会改动原始输入 original input  
      stack.push(...next);  
    } else {  
      res.push(next);  
    }  
  }  
  // 使用 reverse 恢复原数组的顺序  
  return res.reverse();  
}  
flatten(arr1);// [1, 2, 3, 1, 2, 3, 4, 2, 3, 4]
```

5、使用 reduce、concat 和递归无限反嵌套多层嵌套的数组

```
var arr1 = [1,2,3,[1,2,3,4, [2,3,4]]];  
  
function flattenDeep(arr1) {  
  return arr1.reduce((acc, val) => Array.isArray(val) ? acc.concat(flattenDeep(val)) : acc.conca
```

```
t(val), []);  
}  
flattenDeep(arr1);  
// [1, 2, 3, 1, 2, 3, 4, 2, 3, 4]
```

12、怎么判断两个对象相等？（必会）

ES6 中有一个方法判断两个对象是否相等，这个方法判断是两个对象引用地址是否一致

```
let obj1 = {  
  a: 1  
}  
let obj2 = {  
  a: 1  
}  
console.log(Object.is(obj1, obj2)) // false  
let obj3 = obj1  
console.log(Object.is(obj1, obj3)) // true  
console.log(Object.is(obj2, obj3)) // false
```

当需求是比较两个对象内容是否一致时就没用了

想要比较两个对象内容是否一致，思路是要遍历对象的所有键名和键值是否都一致：

- 1、判断两个对象是否指向同一内存
- 2、使用 `Object.getOwnPropertyNames` 获取对象所有键名数组
- 3、判断两个对象的键名数组是否相等
- 4、遍历键名，判断键值是否都相等

```
let obj1 = {  
  a: 1,  
  b: {  
    c: 2  
  }  
}  
let obj2 = {  
  b: {
```

```

        c: 3
    },
    a: 1
}
function isObjectValueEqual(a, b) {
    // 判断两个对象是否指向同一内存，指向同一内存返回 true
    if (a === b) return true
    // 获取两个对象键值数组
    let aProps = Object.getOwnPropertyNames(a)
    let bProps = Object.getOwnPropertyNames(b)
    // 判断两个对象键值数组长度是否一致，不一致返回 false
    if (aProps.length !== bProps.length) return false
    // 遍历对象的键值
    for (let prop in a) {
        // 判断 a 的键值，在 b 中是否存在，不存在，返回 false
        if (b.hasOwnProperty(prop)) {
            // 判断 a 的键值是否为对象，是则递归，不是对象直接判断键值是否相等，不相等返回 false
            if (typeof a[prop] === 'object') {
                if (!isObjectValueEqual(a[prop], b[prop])) return false
            } else if (a[prop] !== b[prop]) {
                return false
            }
        } else {
            return false
        }
    }
    return true
}
console.log(isObjectValueEqual(obj1, obj2)) // false

```

13、列举三种强制类型转换和两种隐式类型转换？（必会）

强制

转化成字符串 toString() String()

转换成数字 Number()、 parseInt()、 parseFloat()

转换成布尔类型 Boolean()

隐式

拼接字符串

例子 `var str = "" + 18`

`- * / % ==`

14、JavaScript 中怎么获取当前日期的月份？（必会）

方法

JavaScript 中获得当前日期是使用 `new Date` 这个内置对象的实例，其他一些进阶的操作也是基于这个内置对象的实例。

获取完整的日期（默认格式）：

```
var date = new Date(); // Sat Jul 06 2019 19:59:27 GMT+0800 (中国标准时间)
```

获取当前年份：

```
var year = date.getFullYear(); // 2019
```

获取当前月份：

```
var month = date.getMonth() + 1; // 7
```

获取当前日：

```
var day = date.getDay(); // 6
```

获取当前日期（年-月-日）：

```
month = (month > 9) ? month : ("0" + month);
```

```
day = (day < 10) ? ("0" + day) : day; var today = year + "-" + month + "-" + day; //
```

2019-07-06

另外的一些操作:

`date.getFullYear();` // 获取当前年份(2 位)

`date.getFullYear();` // 获取完整的年份(4 位, 1970-????)

`date.getMonth();` // 获取当前月份(0-11,0 代表 1 月)

`date.getDate();` // 获取当前日(1-31)

`date.getDay();` // 获取当前星期 X(0-6,0 代表星期天)

`date.getTime();` // 获取当前时间(从 1970.1.1 开始的毫秒数)

`date.getHours();` // 获取当前小时数(0-23)

`date.getMinutes();` // 获取当前分钟数(0-59)

`date.getSeconds();` // 获取当前秒数(0-59)

`date.getMilliseconds();` // 获取当前毫秒数(0-999)

`date.toLocaleDateString();` // 获取当前日期

`date.toLocaleTimeString();` // 获取当前时间

`date.toLocaleString();` // 获取日期与时间

15、什么是类数组（伪数组），如何将其转化为真实的数组？（必会）

伪数组

- 1、具有 `length` 属性
- 2、按索引方式存储数据
- 3、不具有数组的 `push.pop` 等方法

伪数组（类数组）：无法直接调用数组方法或期望 length 属性有什么特殊的行为，不具有数组的 push.pop 等方法，但仍可以对真正数据遍历方法来遍历它们。典型的是函数 document.childNodes 之类的，它们返回的 nodeList 对象都属于伪数组

伪数组-->真实数组

1.使用 Array.from()--ES6

2 [].slice.call(eleArr) 或则 Array.prototype.slice.call(eleArr)

示例：

```
let eleArr = document.querySelectorAll('li');
Array.from(eleArr).forEach(function(item){
  alert(item);
});

let eleArr = document.querySelectorAll('li');
[].slice.call(eleArr).forEach(function(item){
  alert(item);
});
```

16、如何遍历对象的属性？（必会）

1、遍历自身可枚举的属性（可枚举，非继承属性）Object.keys() 方法

该方法会返回一个由一个给定对象的自身可枚举属性组成的数组，数组中的属性名的排列顺序和使用 for..in 遍历该对象时返回的顺序一致（两者的区别是 for ..in 还会枚举其原型链上的属性）

```
/**Array 对象**/
var arr = ['a','b','c'];
console.log(Object.keys(arr));
// ['0','1','2']

/**Object 对象**/
var obj = {foo:'bar',baz:42};
console.log(Object.keys(obj));
```



```
// ["foo","baz"]
/**类数组 对象 随机 key 排序**/
var anObj = {100:'a',2:'b',7:'c'};
console.log(Object.keys());
// ['2','7','100']
/**getFoo 是一个不可枚举的属性**/
var my_obj = Object.create(
  {},
  { getFoo : { value : function () { return this.foo } } }
);
my_obj.foo = 1;
console.log(Object.keys(my_obj));
// ['foo']
```

2、遍历自身的所有属性(可枚举, 不可枚举, 非继承属性) Object.getOwnPropertyNames()

方法, 该方法返回一个由指定对象的所有自身属性组成的数组(包括不可枚举属性但不包括

Symbol 值作为名称的属性)

```
var arr = ["a", "b", "c"];
console.log(Object.getOwnPropertyNames(arr).sort()); // ["0", "1", "2", "length"]
// 类数组对象
var obj = { 0: "a", 1: "b", 2: "c"};
console.log(Object.getOwnPropertyNames(obj).sort()); // ["0", "1", "2"]
// 使用 Array.forEach 输出属性名和属性值
Object.getOwnPropertyNames(obj).forEach(function(val, idx, array) {
  console.log(val + " -> " + obj[val]);
});
// 输出
// 0 -> a
// 1 -> b
// 2 -> c
//不可枚举属性
var my_obj = Object.create({}, {
  getFoo: {
    value: function() { return this.foo; },
    enumerable: false
  }
});
my_obj.foo = 1;
console.log(Object.getOwnPropertyNames(my_obj).sort()); // ["foo", "getFoo"]
```

3、遍历可枚举的自身属性和继承属性 （可枚举，可继承的属性） for in

遍历对象的属性

```
var obj={
  name: '张三',
  age : '24',
  getAge:function(){
    console.log(this.age);
  }
}
var array ={};
for(var i in obj){
  if(obj.hasOwnProperty(i)&& typeof obj[i] != 'function'){
    array[i] = obj[i];
  }
}
console.log(array);
{name:'张三',age:24}
```

注: hasOwnProperty()方法判断对象是有某个属性(本身的属性，不是继承的属性)

4、遍历所有的自身属性和继承属性

```
(function () {
  var getAllPropertyNames = function (obj) {
    var props = [];
    do {
      props = props.concat(Object.getOwnPropertyNames(obj));
    } while (obj = Object.getPrototypeOf(obj));
    return props;
  }
  var propertyNames = getAllPropertyNames(window);
  alert(propertyNames.length); //276
  alert(propertyNames.join("\n")); //toString 等
})();
```

17、如何使用原生 JavaScript 给一个按钮绑定两个 onclick 事件? (必会)

```
Var btn=document.getElementById( 'btn' );
```

```
//事件监听 绑定多个事件
var btn4 = document.getElementById("btn4");
btn4.addEventListener("click",hello1);
btn4.addEventListener("click",hello2);
function hello1(){
    alert("hello 1");
}
function hello2(){
    alert("hello 2");
}
```

18、JavaScript 中的作用域、预解析与变量声明提升？（必会）

作用域

就是变量的有效范围。 在一定的空间里可以对数据进行读写操作，这个空间就是数据的

作用域

- 1、全局作用域： 最外层函数定义的变量拥有全局作用域，即对任何内部函数来说，都是可以访问的；
 - 2、局部作用域： 局部作用域一般只在固定的代码片段内可访问到，而对于函数外部是无法访问的，最常见的例如函数内部。在 ES6 之前，只有函数可以划分变量的作用域，所以在函数的外面无法访问函数内的变量
 - 3、块级作用域：凡是代码块就可以划分变量的作用域，这种作用域的规则就叫块级作用域
- 块级作用域 函数作用域 词法作用域之间的区别：

3.1) 块级作用域和函数作用域描述的是，什么东西可以划分变量的作用域

3.2) 词法作用域描述的是，变量的查找规则

之间的关系：

- 1、 块级作用域 包含 函数作用域

2、词法作用域 与 块级作用域、函数作用域之间没有任何交集， 他们从两个角度描

述了作用域的规则

ES6 之前 JavaScript 采用的是函数作用域+词法作用域，ES6 js 采用的是块级作用域+词法作用域

预解析

JavaScript 代码的执行是由浏览器中的 JavaScript 解析器来执行的。JavaScript 解析器执行 JavaScript 代码的时候，分为两个过程：预解析过程和代码执行过程

预解析过程：

1.把变量的声明提升到当前作用域的最前面，只会提升声明，不会提升赋值

2.把函数的声明提升到当前作用域的最前面，只会提升声明，不会提升调用

3.先提升 var，在提升 function

JavaScript 的执行过程：

1	// 案例 1
2	var a = 25;
3	function abc() {
4	alert(a);
5	var a = 10;
6	}
7	abc();
8	
9	
10	// 案例 2
11	console.log(a);
12	function a() {
13	console.log('aaaaa');
14	}
15	var a = 1;
16	console.log(a);

变量提升

变量提升：定义变量的时候，变量的声明会被提升到作用域的最上面，变量的赋值不会提升

函数提升：JavaScript 解析器首先会把当前作用域的函数声明提前到整个作用域的最前面

1	// 1、 -----
2	var num = 10;
3	fun();
4	function fun() {
5	console.log(num);
6	var num = 20;
7	}
8	//2、 -----
9	var a = 18;
10	f1();
11	function f1() {
12	var b = 9;
13	console.log(a);
14	console.log(b);
15	var a = '123';
16	}
17	// 3、 -----
18	f1();
19	console.log(c);
20	console.log(b);
21	console.log(a);
22	function f1() {
23	var a = b = c = 9;
24	console.log(a);
25	console.log(b);
26	console.log(c);
27	}

变量声明提升：

使用 var 关键字定义的变量，被称为变量声明；

函数声明提升的特点是，在函数声明的前面，可以调用这个函数

19、变量提升与函数提升的区别？（必会）

变量提升

简单说就是在 JavaScript 代码执行前引擎会先进行预编译，预编译期间会将变量声明与函数

声明提升至其对应作用域的最顶端，函数内声明的变量只会提升至该函数作用域最顶层

当函数内部定义的一个变量与外部相同时，那么函数体内的这个变量就会被上升到最顶端

举例来说：

```
console.log(a); //
```

```
undefined var a = 3;
```

```
//预编译后的代码结构可以看做如下运行顺序 var a; // 将变量 a 的声明提升至最顶
```

```
端，赋值逻辑不提升。 console.log(a);
```

```
// undefined a = 3; // 代码执行到原位置即执行原赋值逻辑
```

函数提升

函数提升只会提升函数声明式写法，函数表达式的写法不存在函数提升

函数提升的优先级大于变量提升的优先级，即函数提升在变量提升之上

20、什么是作用域链？（必会）

作用域链

当代码在一个环境中执行时，会创建变量对象的一个作用域链

由子级作用域返回父级作用域中寻找变量，就叫做作用域链

作用域链中的下一个变量对象来自包含环境，也叫外部环境。而再下一个变量对象则来自

下一个包含环境，一直延续到全局执行环境。全局执行环境的变量对象始终都是作用域链

中的最后一个对象

作用域链前端始终都是当前执行的代码所在环境的变量对象，如果环境是函数，则将其活动对象作为变量对象

21、如何延长作用域链？（必会）

作用域链是可以延长的

延长作用域链：

执行环境的类型只有两种，全局和局部（函数）。但是有些语句可以在作用域链的前端临时增加一个变量对象，该变量对象会在代码执行后被移除

具体来说就是执行这两个语句时，作用域链都会得到加强

1、try - catch 语句的 catch 块；会创建一个新的变量对象，包含的是被抛出的错误对象的声明

2、with 语句。with 语句会将指定的对象添加到作用域链中

22、判断一个值是什么类型有哪些方法？（必会）

方法

1、typeof 运算符

2、instanceof 运算符

instanceof 严格来说是 Java 中的一个双目运算符，用来测试一个对象是否为一个类的实例，用法为：

```
// 判断 foo 是否是 Foo 类的实例
function Foo(){}

```

```
var foo = new Foo();
console.log(foo instanceof Foo) //true
```

3、Object.prototype.toString 方法

在 JavaScript 里使用 `typeof` 来判断数据类型, 只能区分基本类型, 即 “number”, “string”, “undefined”, “boolean”, “object”, “function”, “symbol” (ES6 新增)七种

对于数组、null、对象来说, 其关系错综复杂, 使用 `typeof` 都会统一返回 “object” 字符串

要想区别对象、数组、函数单纯使用 `typeof` 是不行的, JavaScript 中,通过

`Object.prototype.toString` 方法, 判断某个对象值属于哪种内置类型。

在介绍 `Object.prototype.toString` 方法之前, 我们先把 `toString()`方法和

`Object.prototype.toString.call()`方法进行对比

`toString()`方法和 `Object.prototype.toString.call()`方法对比

```
var arr=[1,2];

//直接对一个数组调用 toString()

arr.toString();// "1,2"

//通过 call 指定 arr 数组为 Object.prototype 对象中的 toString 方法的上下文

Object.prototype.toString.call(arr); //"object Array"
```

23、如何实现数组的随机排序? (必会)

方法一:

```
var arr = [1,2,3,4,5,6,7,8,9,10];
function randSort1(arr){
```

```
        for(var i = 0,len = arr.length;i < len; i++ ){
            var rand = parseInt(Math.random()*len);
            var temp = arr[rand];
            arr[rand] = arr[i];
            arr[i] = temp;
        }
        return arr;
    }
    console.log(randSort1(arr));
```

方法二:

```
var arr = [1,2,3,4,5,6,7,8,9,10];
function randSort2(arr){
    var mixedArray = [];
    while(arr.length > 0){
        var randomIndex = parseInt(Math.random()*arr.length);
        mixedArray.push(arr[randomIndex]);
        arr.splice(randomIndex, 1);
    }
    return mixedArray;
}
console.log(randSort2(arr));
```

方法三:

```
var arr = [1,2,3,4,5,6,7,8,9,10];
arr.sort(function(){
    return Math.random() - 0.5;
})
console.log(arr);
```

24、src 和 href 的区别是? (了解)

区别

src (source) 指向外部资源的位置, 指向的内容将会嵌入到文档中当前标签所在位置; 在

请求 src 资源时会将其指向的资源下载并应用到文档中，如 JavaScript 脚本，img 图片和 iframe 等元素

当浏览器解析到该元素时，会暂停其他资源的下载和处理，直到将该资源加载、编译、执行完毕，类似于将所指向资源嵌入当前标签内

href (hypertext reference/超文本引用) 指向网络资源所在位置，建立和当前元素（锚点）或当前文档（链接）之间的链接，如果我们在文档中添加<link href="common.css"rel="stylesheet"/>那么浏览器会识别该文档为 CSS 文件，就会并行下载资源并且不会停止对当前文档的处理

WebAPI

1、什么是 dom?（必会）

什么是 dom

- 1、DOM 是 W3C（万维网联盟）的标准
- 2、DOM 定义了访问 HTML 和 XML 文档的标准

什么是 W3C

- 1、“W3C 文档对象模型（DOM）是中立于平台和语言的接口，它允许程序和脚本动态地访问和更新文档的内容、结构和样式。”
- 2、W3C DOM 标准被分为 3 个不同的部分
 - 2.1) 核心 DOM - 针对任何结构化文档的标准模型
 - 2.2) XML DOM - 针对 XML 文档的标准模型
 - 2.3) HTML DOM - 针对 HTML 文档的标准模型

备注: DOM 是 Document Object Model (文档对象模型) 的缩写

2、dom 节点的 Attribute 和 Property 有何区别? (必会)

1、什么是 Property

每个 DOM 节点都是一个 object 对象, 它可以像其他的 js Object 一样具有自己的 property 和 method, 所以 property 的值可以是任何数据类型, 大小写敏感, 原则上 property 应该仅供 js 操作, 不会出现在 html 中 (默认属性除外: id/src/href/className/dir/title/lang 等), 和其他 js object 一样, 自定义的 property 也会出现在 object 的 for...in 遍历中

2、什么是 Attribute

attribute 出现在 dom 中, js 提供了 getAttribute/setAttribute 等方法来获取和改变它的值, attribute 的值只能是字符串且大小写不敏感, 最后作用于 html 中, 可以影响 innerHTML 获取的值。可以通过访问 dom 节点的 attributes 属性来获取 改节点的所有的 attribute。(在 IE<9 中,attribute 获取和改变的实际上是 property。)

3、两者之间的区别是:

3.1) 自定义的 Property 与 Attribute 不同步,不相等

3.2) 非自定义的 DOM property 与 attributes 是有条件同步的

3.3) 非自定义的属性(id/src/href/name/value 等), 通过 setAttribute 修改其特性值可以同步作用到 property 上, 而通过.property 修改属性值有的(value)时候不会同步到 attribute 上, 即不会反应到 html 上(除以下几种情况, 非自定义属性 在二者之间是同步的)。

3、dom 结构操作怎样添加、移除、移动、复制、创建和查找节点？ (必会)

1、创建新节点

`createDocumentFragment()` //创建一个 DOM 片段

`createElement()` //创建一个具体的元素

`createTextNode()` //创建一个文本节点

2、添加、移除、替换、插入

`appendChild()`

`removeChild()`

`replaceChild()`

`insertBefore()` //并没有 `insertAfter()`

3、查找

`getElementsByTagName()` //通过标签名称

`getElementsByName()` //通过元素的 Name 属性的值(IE 容错能力较强，

会得到一个数组，其中包括 id 等于 name 值的)

`getElementById()` //通过元素 Id，唯一性

4、dom 事件模型？ (必会)

DOM 事件模型。

DOM 事件模型分为两种：事件捕获和事件冒泡。

事件捕获以点击事件为例，同类型事件会由根元素开始触发，向内传播，一直到目标元素。

从外到内依次触发：根—目标的祖先素—目标的父元素—目标元素

事件冒泡和事件捕获截然相反。发生点击事件时，事件会从目标元素上开始触发，向外传播，一直到根元素停止。从内到外依次触发：目标元素—目标元素的父元素—父元素的父元素—根

事件传播

事件捕获和事件冒泡都有事件传播阶段，传播阶段就是事件从触发开始到结束的过程。

优先级：先捕获，再冒泡。

两种传播方式的来源：W3C 推行 DOM2 级事件之前网景和 IE 在打架，网景用的事件传播方式是捕获，IE 用的事件传播方式是冒泡

5、什么是事件冒泡，它是如何工作的？如何阻止事件冒泡、默认为？(必会)

1、什么是事件冒泡，他是如何工作的

在一个对象上触发某类事件（比如单击 onclick 事件），如果此对象定义了此事件的处理程序，那么此事件就会调用这个处理程序，如果没有定义此事件处理程序或者事件返回 true，那么这个事件会向这个对象的父级对象传播，从里到外，直至它被处理（父级对象所有同类事件都将被激活），或者它到达了对象层次的最顶层，即 document 对象（有些浏览器是 window）

2、阻止事件冒泡的方法

2.1) w3c 方法是: `event.stopPropagation()`; 事件处理过程中, 阻止冒泡事件, 但不会阻止默认行为 (跳转至超链接)

2.2) IE 则是使用 `event.cancelBubble = true` 阻止事件冒泡

2.3) `return false`; jq 里面事件处理过程中, 阻止冒泡事件, 也阻止默认行为 (不跳转超链接)

封装方法:

```
function bubbles(e){  
  
    var ev = e || window.event;  
  
    if(ev && ev.stopPropagation) {  
  
        //非 IE 浏览器  
  
        ev.stopPropagation();  
  
    } else {  
  
        //IE 浏览器(IE11 以下)  
  
        ev.cancelBubble = true;  
  
    }  
  
    console.log("最底层盒子被点击了")  
  
}
```

阻止默认行为:

w3c 的方法是 `e.preventDefault()`, IE 则是使用 `e.returnValue = false`;

封装:

```
//假定有链接<a href="http://caibaojian.com/" id="testA" >caibaojian.com</a>  
var a = document.getElementById("testA");
```

```
a.onclick =function(e){
if(e.preventDefault){
e.preventDefault();
}else{
window.event.returnValue == false;
}
}
```

6、JavaScript 动画和 CSS3 动画有什么区别？（必会）

1、CSS 动画

优点：

1.1)浏览器可以对动画进行优化。

1.1.1)浏览器使用与 requestAnimationFrame 类似的机制，

requestAnimationFrame 比起 setTimeout，setInterval 设置动画的优势主要

是:1)requestAnimationFrame 会把每一帧中的所有 DOM 操作集中起来，在一次重

绘或回流中就完成,并且重绘或回流的时间间隔紧紧跟随浏览器的刷新频率,一般来说,

这个频率为每秒 60 帧。2)在隐藏或不可见的元素中 requestAnimationFrame 不会进

行重绘或回流，这当然就意味着更少的的 cpu，gpu 和内存使用量。

1.1.2)强制使用硬件加速 （通过 GPU 来提高动画性能）

1.2)代码相对简单,性能调优方向固定

1.3)对于帧速表现不好的低版本浏览器，CSS3 可以做到自然降级，而 JS 则需要撰写

额外代码

缺点：

1.1) 运行过程控制较弱,无法附加事件绑定回调函数。CSS 动画只能暂停,不能在动画中寻找一个特定的时间点,不能在半路反转动画,不能变换时间尺度,不能在特定的位置添加回调函数或是绑定回放事件,无进度报告。

1.2) 代码冗长。想用 CSS 实现稍微复杂一点动画,最后 CSS 代码都会变得非常笨重。

2、JS 动画

优点:

2.1) JavaScript 动画控制能力很强,可以在动画播放过程中对动画进行控制:开始、暂停、回放、终止、取消都是可以做到的。

2.2) 动画效果比 css3 动画丰富,有些动画效果,比如曲线运动,冲击闪烁,视差滚动效果,只有 JavaScript 动画才能完成。

2.3) CSS3 有兼容性问题,而 JS 大多时候没有兼容性问题。

缺点:

2.1) JavaScript 在浏览器的主线程中运行,而主线程中还有其它需要运行的 JavaScript 脚本、样式计算、布局、绘制任务等,对其干扰导致线程可能出现阻塞,从而造成丢帧的情况。

2.2) 代码的复杂度高于 CSS 动画

总结:如果动画只是简单的状态切换,不需要中间过程控制,在这种情况下,css 动画是优选方案。它可以让你将动画逻辑放在样式文件里面,而不会让你的页面充斥 Javascript 库。然而如果你在设计很复杂的富客户端界面或者在开发一个有着复杂 UI 状态的 APP。那么你应该使用 js 动画,这样你的动画可以保持高效,并且你的工作流

也更可控。所以，在实现一些小的交互动效的时候，就多考虑考虑 CSS 动画。对于一些复杂控制的动画，使用 javascript 比较可靠。

3、css 动画和 js 动画的差异

3.1) 代码复杂度，js 动画代码相对复杂一些。

3.2) 动画运行时，对动画的控制程度上，js 能够让动画，暂停，取消，终止，css 动画不能添加事件。

3.3) 动画性能看，js 动画多了一个 js 解析的过程，性能不如 css 动画好。

7、event 对象的常见应用？（必会）

1、event.preventDefault(); // 阻止默认行为，阻止 a 链接默认的跳转行为

2、event.stopPropagation(); // 阻止冒泡

3、event.stopImmediatePropagation(); // 按钮绑定了 2 个响应函数，依次注册 a,b

两个事件，点击按钮，a 事件中加 event.stopImmediatePropagation()就能阻止 b 事件

4、event.currentTarget // 早期的 ie 不支持，当前绑定的事件

5、event.target

8、通用事件绑定/ 编写一个通用的事件监听函数？（必会）

```
function bindEvent(elem, type, selector, fn) {  
  if (fn == null) {  
    fn = selector;  
    selector = null;  
  }  
  elem.addEventListener(type, function(e) {
```

```
        var target;
        if (selector) {
            target = e.target;
            if (target.matches(selector)) {
                fn.call(target, e);
            }
        } else {
            fn(e);
        }
    })
}
// 使用代理
var div1 = document.getElementById('div1');
bindEvent(div1, 'click', 'a', function(e) {
    console.log(this.innerHTML);
});
// 不使用代理
var a = document.getElementById('a1');
bindEvent(div1, 'click', function(e) {
    console.log(a.innerHTML);
})
```

1、代理的好处

1.2) 代码简洁

1.2) 减少浏览器内存占用

2、事件冒泡

事件冒泡的应用：代理

9、DOM 和 BOM 的区别（必会）

1、BOM

1.1) BOM 是 Browser Object Model 的缩写，即浏览器对象模型。

1.2) BOM 没有相关标准。

1.3) BOM 的最根本对象是 window

2、DOM

2.1) DOM 是 Document Object Model 的缩写，即文档对象模型。

2.2) DOM 是 W3C 的标准。

2.3) DOM 最根本对象是 document（实际上是 window.document）

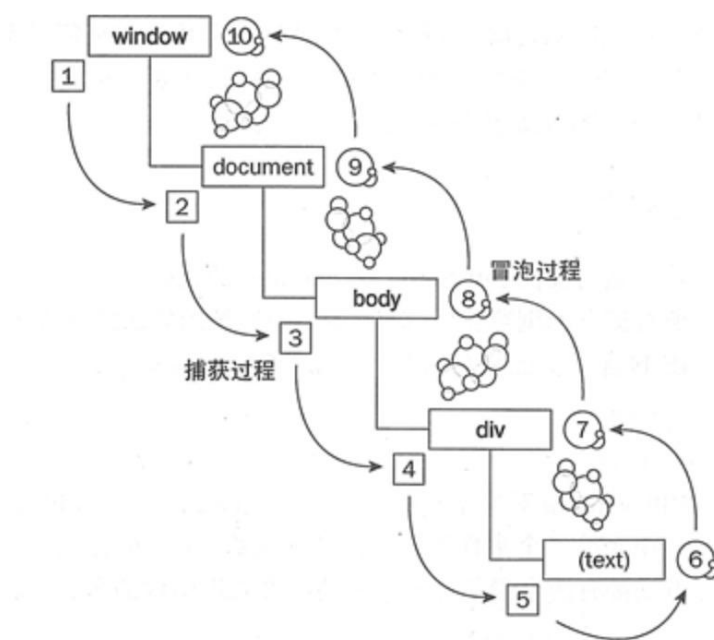
10、事件三要素（必会）

1、事件源、就是你点的那个 div，触发的对象

2、事件类型、表示动作，比如点击、滑过等

3、事件处理函数（事件处理程序）、表示结果，比如点开关跳转到另一个页面

11、事件执行过程（必会）



事件捕获过程：当我们点击 TEXT 时，首先是 window->document->body->div->text.

这个过程称为事件捕获，W3C 浏览器的标准执行流程。

事件冒泡过程：text->div->body->document->window.这个过程称为事件冒泡。IE 浏览器只支持冒泡，不支持捕获。W3C 浏览器先执行捕获，后执行冒泡

12、获取元素位置（必会）

1、通过元素的 offsetLeft 和 offsetTop

dom 元素的 offsetLeft、offsetTop 指的是元素相对于其 offsetParent 指定的坐标来说的。offsetParent：是指当前元素最近的经过定位的父级元素，如果没有则一直向上直至 body。注意当前元素为 fixed 时，其 offsetParent 的值为 null

拓展：

offsetWidth/offsetHeight: width+padding+border

clientLeft/clientTop:表示内容区域的左上角相对于整个元素左上角的位置（包括边框）//

个人理解为 border 值

clientWidth/clientHeight: width+padding

scrollWidth:获取对象的滚动宽度

scrollHeight: 获取对象的滚动高度。

scrollLeft:设置或获取位于对象左边界和窗口中目前可见内容的最左端之间的距离

scrollTop:设置或获取位于对象最顶端和窗口中可见内容的最顶端之间的距离

window.screen.availHeight/window.screen.availWidth: 浏览器去除上方工具栏和下方菜单栏可用宽高

window.screen.height/window.screen.width: 屏幕宽高

2、event.clientX 和 event.clientY

事件相对于文档的水平和垂直距离

3、getBoundingClientRect

方法返回一个矩形对象，包含四个属性：left、top、right 和 bottom。分别表示元素各边与页面上边和左边的距离

13、封装运动函数（必会）

```
/*
obj 指的是 DOM 对象
- json 指的是 CSS 样式
例 startMove(oDiv,{width:100,height:100},function(){})
*/
function startMove(obj,json,fnEnd){
    clearInterval(obj.timer); //先清除之前的定时器
    obj.timer = setInterval(function(){
        var bStop = true; // 假设所有的值都到了
        for( var attr in json ){ //遍历 json 属性
            var cur = (attr == 'opacity') ? Math.round(parseFloat(getStyle(obj,attr))*100) :
parseFloat(getStyle(obj,attr)); //对 opacity 特殊处理
            var speed = (json[attr] - cur)/6;
            speed = speed > 0 ? Math.ceil(speed) : Math.floor(speed); //speed 数字
            转化，防止不能到达目标的 bug
            if( cur != json[attr] ) bStop = false; //如果没有达到目标值，则 bStop 设为 false;
            if(attr == 'opacity'){
                obj.style.filter = 'alpha(opacity='+ (cur + speed) + ')';
                obj.style.opacity = (cur + speed)/100;
            }else{
                obj.style[attr] = cur + speed + 'px';
            }
        }
    }, 10);
    if(bStop){
        clearInterval(obj.timer);
        if(fnEnd) fnEnd(); //执行回调函数
    }
}
```

```

    }
    },30);
}
function getStyle(obj,name){
    return obj.currentStyle ? obj.currentStyle[name] :
window.getComputedStyle(obj,null)[name]; //浏览器兼容性处理, 注意 getComputedStyle 为只读属性
}

function getByClass(oParent,sClass){
    var aEle = oParent.getElementsByTagName("*");
    var aResult = [];
    var re = new RegExp("\\b" + sClass + "\\b','i');
    for(var i=0; i<aEle.length;i++ ){
        if(re.test(aEle[i].className)) aResult.push(aEle[i]);
    }
    return aResult;
}

```

14、绑定事件和解除事件的区别（必会）

1、事件绑定

定义：一个事件可以加多次，且不会覆盖

2、绑定方法

2.1) attachEvent ('on+事件名', 函数名) 这个只兼容 ie 6-8

2.2) addEventListener (事件名, 函数名, false) 支持 ie9+ chrom firfox

绑定事件的封装

```

function addEvent(obj,sEv,fn){
    if(obj.addEventListener){
        obj.addEventListener(sEv,fn,false);
    }else{
        obj.attachEvent('on'+sEv,fn);
    }
};

```

解除绑定事件的封装

```
function removeEvent(obj,sEv,fn){
    if(obj.removeEventListener){
        obj.removeEventListener(sEv,fn,false);
    }else{
        obj.detachEvent('on'+sEv,fn);
    }
}
```

15、谈谈事件委托的理解？（必会）

JavaScript 事件代理则是一种简单的技巧,通过它你可以把事件处理器添加到一个上级元素上,这样就避免了把事件处理器添加到多个子级元素上。当我们需要对很多元素添加事件的时候,可以通过将事件添加到它们的上级元素而将事件委托给上级元素来触发处理函数。这主要得益于浏览器的事件冒泡机制。事件代理用到了两个在 JavaScript 事件中常被忽略的特性:事件冒泡以及目标元素。

优点:

- 1、减少事件注册,节省内存。
- 2、在 table 上代理所有 td 的 click 事件。
- 3、在 ul 上代理所有 li 的 click 事件。
- 4、简化了 dom 节点更新时,相应事件的更新。
- 5、不用在新添加的 li 上绑定 click 事件。
- 6、当删除某个 li 时,不用移解绑上面的 click 事件。

缺点:

-
- 1、事件委托基于冒泡，对于不冒泡的事件不支持
 - 2、层级过多，冒泡过程中，可能会被某层阻止掉。
 - 3、理论上委托会导致浏览器频繁调用处理函数，虽然很可能不需要处理。所以建议就近委托，比如在 table 上代理 td，而不是在 document 上代理 td。
 - 4、把所有事件都用代理就可能会出现事件误判。比如，在 document 中代理了所有 button 的 click 事件，另外的人在引用改 js 时，可能不知道，造成单击 button 触发了两个 click 事件

16、JavaScript 中的定时器有哪些？他们的区别及用法是什么？（必会）

1、JavaScript 中的定时器有以下几种

- 1) setTimeout() 方法用于在指定的毫秒数后调用函数或计算表达式。
- 2) setInterval() 方法可按照指定的周期（以毫秒计）来调用函数或计算表达式。

setInterval() 方法会不停地调用函数，直到 clearInterval() 被调用或窗口被关闭。由 setInterval() 返回的 ID 值可用作 clearInterval() 方法的参数。

17、比较 attachEvent 和 addEventListener?（必会）

attachEvent 方法可以动态的为网页内的元素添加一个事件。通常你想为某个按钮添加一个单击事件时，你都会在按钮内写上 onclick=事件名称。使用 attachEvent 则不必这样做。你把写好的事件准备好，在需要的时候给元素绑定上再执行。而且 attachEvent 支持为某个元素绑定多个事件。执行顺序是，后绑定的先执行。如果想删除事件请使用 detachEvent

attachEvent 方法只支持 IE 浏览器。与其功能相同的指令是 addEventListener,该指令支持 FF 等浏览器，并且是 W3C 标准

语法：Element.attachEvent(Etype,EventName)

参数 Element:要为该元素动态添加一个事件

Etype:指定事件类型。比如：onclick,onkeyup,onmousemove 等

EventName:指定事件名称。也就是你写好的函数

addEventListener 方法与 attachEvent 方法功能相同。但是 addEventListener 是 W3C 标准，而 attachEvent 非 W3C 标准，且只支持 IE 浏览器

虽然 addEventListener 属于标准方法，但依然无法在 IE 下运行。IE 不支持该方法

addEventListener 带有三个参数必须设置缺一不可

addEventListener 可以为网页内某个元素动态绑定一个事件。事件类型可随意指定。如:click,mousemove,keyup 等

通常你想为某个按钮添加一个单击事件时。你都会在按钮内写上onclick=事件名称.使用

addEventListener 则不必这样做。你把写好的事件准备好，在需要的时候给元素绑定上再执行。而且 addEventListener 支持为某个元素绑定多个事件。执行顺序是，先绑定的先执行。如果想删除事件请使用 removeEventListener

语法：Element.addEventListener(Etype,EventName,boole)返回值：[tag:return_value /]

参数 Element:要为该元素绑定一个事件。可以是任意的 html 元素

Etype:事件类型.比如：click,keyup,mousemove.注意使用 addEventListener 绑定事件时，设置参数事件类型时不必写 on。否则会出错

eventName:要绑定事件的名称。也就是你写好的函数

bool:该参数是一个布尔值: false 或 true 必须填写。false 代表支持浏览器事件捕获功能, true 代表支持浏览事件冒泡功能

18、document.write 和 innerHTML 的区别? (必会)

document.write 是直接写入到页面的内容流,如果在写之前没有调用 document.open, 浏览器会自动调用 open。每次写完关闭之后重新调用该函数, 会导致页面被重写

innerHTML 则是 DOM 页面元素的一个属性, 代表该元素的 html 内容。你可以精确到某一个具体的元素来进行更改。如果想修改 document 的内容, 则需要修改

document.documentElement.innerHTML

innerHTML 将内容写入某个 DOM 节点, 不会导致页面全部重绘

innerHTML 很多情况下都优于 document.write, 其原因在于其允许更精确的控制要刷新页面的那一个部分

19、什么是 window 对象? 什么是 document 对象? (必会)

1、什么是 window 对象

简单来说, document 是 window 的一个对象属性。

Window 对象表示浏览器中打开的窗口。

如果文档包含框架 (frame 或 iframe 标签), 浏览器会为 HTML 文档创建一个 window 对象, 并为每个框架创建一个额外的 window 对象。

所有的全局函数和对象都属于 Window 对象的属性和方法。

document 对 Document 对象的只读引用。

[window 对象]

它是一个顶层对象,而不是另一个对象的属性,即浏览器的窗口。

属性

defaultStatus 缺省的状态条消息

document 当前显示的文档(该属性本身也是一个对象)

frame 窗口里的一个框架((FRAME>))(该属性本身也是一个对象)

frames array 列举窗口的框架对象的数组,按照这些对象在文档中出现的顺序列出
(该属性本身也是一个对象)

history 窗口的历史列表(该属性本身也是一个对象)

length 窗口内的框架数

location 窗口所显示文档的完整(绝对)URL(该属性本身也是一个对象)不要把它与
如 document.location 混淆,后者是当前显示文档的 URL。用户可以改变 window.location(用
另一个文档取代当前文档),但却不能改变 document.location (因为这是当前显示文档的位置)

name 窗口打开时,赋予该窗口的名字

opener 代表使用 window.open 打开当前窗口的脚本所在的窗口(这是 Netscape
Navigator 3.0beta 3 所引入的一个新属性)

parent 包含当前框架的窗口的同义词。frame 和 window 对象的一个属性

self 当前窗口或框架的同义词

status 状态条中的消息

top 包含当前框架的最顶层浏览器窗口的同义词

window 当前窗口或框架的同义词,与 self 相同

方法

alert() 打开一个 Alert 消息框

clearTimeout() 用来终止 setTimeout 方法的工作

close() 关闭窗口

confirm() 打开一个 Confirm 消息框,用户可以选择 OK 或 Cancel,如果用户单击

OK,该方法返回 true,单击 Cancel 返回 false

blur() 把焦点从指定窗口移开(这是 Netscape Navigator 3.0 beta 3 引入的新方法)

focus() 把指定的窗口带到前台(另一个新方法)

open() 打开一个新窗口

prompt() 打开一个 Prompt 对话框,用户可向该框键入文本,并把键入的文本返回到脚本

setTimeout() 等待一段指定的毫秒数时间,然后运行指令事件处理程序事件处理程序

Onload() 页面载入时触发

Onunload() 页面关闭时触发

2、什么是 document 对象

[document 对象]

该对象是 window 和 frames 对象的一个属性,是显示于窗口或框架内的一个文档。

属性

alinkColor 活动链接的颜色(ALINK)

anchor 一个 HTML 锚点,使用标记创建(该属性本身也是一个对象)

anchors array 列出文档锚点对象的数组()(该属性本身也是一个对象)

bgColor 文档的背景颜色(BGCOLOR)

cookie 存储于 cookie.txt 文件内的一段信息,它是该文档对象的一个属性

fgColor 文档的文本颜色(<BODY> 标记里的 TEXT 特性)

form 文档中的一个窗体(<FORM>)(该属性本身也是一个对象)

forms array 按照其出现在文档中的顺序列出窗体对象的一个数组(该属性本身也是一个对象)

lastModified 文档最后的修改日期

linkColor 文档的链接的颜色,即<BODY>标记中的 LINK 特性(链接到用户没有观察到的文档)

link 文档中的一个标记(该属性本身也是一个对象)

links array 文档中 link 对象的一个数组,按照它们出现在文档中的顺序排列(该属性本身也是一个对象)

location 当前显示文档的 URL。用户不能改变 document.location(因为这是当前显示文档的位置)。但是,可以改变 window.location (用其它文档取代当前文档)window.location 本身也是一个对象,而 document.location 不是对象

referrer 包含链接的文档的 URL,用户单击该链接可到达当前文档

title 文档的标题((TITLE>)

vlinkColor 指向用户已观察过的文档的链接文本颜色,即<BODY>标记的 VLINK 特性

方法

clear 清除指定文档的内容

close 关闭文档流

open 打开文档流

write 把文本写入文档

writeln 把文本写入文档,并以换行符结尾

区别:1、 window 指窗体。 document 指页面。 document 是 window 的一个子对象。

2、用户不能改变 document.location(因为这是当前显示文档的位置)。但是,可以改变 window.location (用其它文档取代当前文档)window.location 本身也是一个对象,而 document.location 不是对象

20、Js 拖动的原理? (必会)

js 的拖拽效果主要用到以下三个事件:

mousedown 鼠标按下事件

mousemove 鼠标移动事件

mouseup 鼠标抬起事件

当点击 dom 的时候,记录当前鼠标的坐标值,也就是 x、y 值,以及被拖拽的 dom 的 top、left 值,而且还要在鼠标按下的回调函数里添加鼠标移动的事件:

document.addEventListener("mousemove", moving, false)和添加鼠标抬起的事件

```
document.addEventListener("mouseup",function()
```

```
{ document.removeEventListener("mousemove", moving, false);}, false);
```

这个抬起的事件是为了解除鼠标移动的监听，因为只有在鼠标按下才可以拖拽，抬起就停止不会移动了。

当鼠标按下鼠标移动的时候，记录移动中的 x、y 值，那么这个被拖拽的 dom 的 top 和 left 值就是：

top=鼠标按下时记录的 dom 的 top 值+（移动中的 y 值 - 鼠标按下时的 y 值）

left=鼠标按下时记录的 dom 的 left 值+（移动中的 x 值 - 鼠标按下时的 x 值）；

```
<div id="drabble"></div>
<script>
  window.onload = function() {
    var dom = document.getElementById("drabble");
    dom.addEventListener(
      "mousedown",
      function(event) {
        var x = event.clientX;
        var y = event.clientY;
        var marginLeft = parseInt(dom.offsetLeft);
        var marginTop = parseInt(dom.offsetTop);
        function moving(e) {
          var movedX = e.clientX - x;
          var movedY = e.clientY - y;
          dom.style.marginLeft = marginLeft + movedX + "px";
          dom.style.marginTop = marginTop + movedY + "px";
        }
        document.addEventListener("mousemove", moving, false);
        document.addEventListener("mouseup",function() {
          document.removeEventListener("mousemove", moving, false);
        }, false);
      },
      false
    );
  };
</script>
```

21、描述浏览器的渲染过程，DOM 树和渲染树的区别（必会）

1、浏览器的渲染过程：

解析 HTML 构建 DOM(DOM 树)，并行请求 css/image/js

CSS 文件下载完成，开始构建 CSSOM(CSS 树)

CSSOM 构建结束后，和 DOM 一起生成 Render Tree(渲染树)

布局(Layout)：计算出每个节点在屏幕中的位置

显示(Painting): 通过显卡把页面画到屏幕上

2、DOM 树 和 渲染树 的区别

DOM 树与 HTML 标签一一对应, 包括 head 和隐藏元素

渲染树不包括 head 和隐藏元素, 大段文本的每一个行都是独立节点, 每一个节点都有对应的 css 属性

22、如何最小化重绘(repaint)和回流(reflow) (必会)

什么是重绘 Repaint 和重排 (回流 reflow)

重绘:当元素的一部分属性发生改变, 如外观、背景、颜色等不会引起布局变化, 只需要浏览器根据元素的新属性重新绘制, 使元素呈现新的外观叫做重绘。

重排 (回流):当 render 树中的一部分或者全部因为大小边距等问题发生改变而需要 DOM 树重新计算的过程

重绘不一定需要重排 (比如颜色的改变), 重排必然导致重绘 (比如改变网页位置)

方法:

需要要对元素进行复杂的操作时, 可以先隐藏(display:"none"), 操作完成后再显示

需要创建多个 DOM 节点时, 使用 DocumentFragment 创建完后一次性的加入 document

缓存 Layout 属性值, 如: var left = elem.offsetLeft; 这样, 多次使用 left 只产生一次回流

尽量避免用 table 布局 (table 元素一旦触发回流就会导致 table 里所有的其它元素回流)

避免使用 css 表达式(expression), 因为每次调用都会重新计算值 (包括加载页面)

尽量使用 css 属性简写, 如: 用 border 代替 border-width, border-style, border-color

批量修改元素样式: `elem.className` 和 `elem.style.cssText` 代替 `elem.style.xxx`

23、简单说一下页面重绘和回流？（高薪常问）

1、什么是回流

回流：当 render tree 的一部分或全部的元素因改变了自身的宽高，布局，显示或隐藏，或者元素内部的文字结构发生变化 导致需要重新构建页面的时候，回流就产生了

2、什么是重绘

重绘：当一个元素自身的宽高，布局，及显示或隐藏没有改变，而只是改变了元素的外观风格的时候，就会产生重绘。例如你改变了元素的 `background-color....`因此得出了一个结论：回流必定触发重绘，而重绘不一定触发回流

24、Js 延迟加载的方式有哪些？（了解）

js 实现延迟加载的几种方法，js 的延迟加载有助于提高页面的加载速度

JS 延迟加载，也就是等页面加载完成之后再加载 JavaScript 文件

JS 延迟加载有助于提高页面加载速度

一般有以下几种方式：

`defer` 属性

`async` 属性

动态创建 DOM 方式

使用 jQuery 的 `getScript` 方法

使用 `setTimeout` 延迟方法

让 JS 最后加载

1、defer 属性

HTML 4.01 为<script>标签定义了 defer 属性。标签定义了 defer 属性元素中设置 defer 属性，等于告诉浏览器立即下载，但延迟执行标签定义了 defer 属性。

用途：表明脚本在执行时不会影响页面的构造。也就是说，脚本会被延迟到整个页面都解析完毕之后再执行

在<script>元素中设置 defer 属性，等于告诉浏览器立即下载，但延迟执行

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <script src="test1.js" defer="defer"></script>
5     <script src="test2.js" defer="defer"></script>
6 </head>
7 <body>
8 <!-- 这里放内容 -->
9 </body>
10 </html>
```

说明：虽然<script>元素放在了<head>元素中，但包含的脚本将延迟浏览器遇到</html>标签后再执行

HTML5 规范要求脚本按照它们出现的先后顺序执行。在现实当中，延迟脚本并不一定会按照顺序执行

defer 属性只适用于外部脚本文件。支持 HTML5 的实现会忽略嵌入脚本设置的 defer 属性

2、async 属性

HTML5 为<script>标签定义了 async 属性。与 defer 属性类似，都用于改变处理脚本的行为。同样，只适用于外部脚本文件。标签定义了 async 属性。与 defer 属性类似，都用于改变处理脚本的行为。同样，只适用于外部脚本文件

目的：不让页面等待脚本下载和执行，从而异步加载页面其他内容。异步脚本一定会

在页面 load 事件前执行。不能保证脚本会按顺序执行

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <script src="test1.js" async></script>
5     <script src="test2.js" async></script>
6 </head>
7 <body>
8 <!-- 这里放内容 -->
9 </body>
10 </html>
```

async 和 defer 一样，都不会阻塞其他资源下载，所以不会影响页面的加载。

缺点：不能控制加载的顺序

3、动态创建 DOM 方式

```
1 //这些代码应被放置在</body>标签前(接近HTML文件底部)
2 <script type="text/javascript">
3     function downloadJSAtOnload() {
4         varelement = document.createElement("script");
5         element.src = "defer.js";
6         document.body.appendChild(element);
7     }
8     if (window.addEventListener)
9         window.addEventListener("load",downloadJSAtOnload, false);
10    else if (window.attachEvent)
11        window.attachEvent("onload",downloadJSAtOnload);
12    else
13        window.onload =downloadJSAtOnload;
14 </script>
```

4、使用 jQuery 的 getScript()方法

```
1 $.getScript("outer.js",function(){//回调函数，成功获取文件后执行的函数
2     console.log("脚本加载完成")
3 });
```

5、使用 setTimeout 延迟方法的加载时间

延迟加载 js 代码，给网页加载留出更多时间

```

1 <script type="text/javascript" >
2     function A(){
3         $.post("/lord/login",{name:username,pwd:password},function(){
4             alert("Hello");
5         });
6     }
7     $(function (){
8         setTimeout('A()', 1000); //延迟1秒
9     })
10 </script>

```

6、让 JS 最后加载

把 js 外部引入的文件放到页面底部，来让 js 最后引入，从而加快页面加载速度

例如引入外部 js 脚本文件时，如果放入 html 的 head 中，则页面加载前该 js 脚本就会被加载入页面，而放入 body 中，则会按照页面从上倒下的加载顺序来运行 JavaScript 的代码

所以我们可以把 js 外部引入的文件放到页面底部，来让 js 最后引入，从而加快页面加载速度

上述方法 2 也会偶尔让你收到 Google 页面速度测试工具的“延迟加载 javascript”

警告。所以这里的解决方案将是来自 Google 帮助页面的推荐方案。

```

1 //这些代码应被放置在</body>标签前(接近HTML文件底部)
2 <script type="text/javascript">
3     function downloadJSAtOnload() {
4         var element = document.createElement("script");
5         element.src = "defer.js";
6         document.body.appendChild(element);
7     }
8     if (window.addEventListener)
9         window.addEventListener("load", downloadJSAtOnload, false);
10    else if (window.attachEvent)
11        window.attachEvent("onload", downloadJSAtOnload);
12    else window.onload = downloadJSAtOnload;
13 </script>

```

这段代码意思等到整个文档加载完后，再加载外部文件“defer.js”。

使用此段代码的步骤：

6.1) 复制上面代码

6.2) 粘贴代码到 HTML 的标签前（靠近 HTML 文件底部）

6.3) 修改 “defer.js” 为你的外部 JS 文件名

6.4) 确保你文件路径是正确的。例如：如果你仅输入 “defer.js” ，那么 “defer.js”

文件一定与 HTML 文件在同一文件夹下。

注意：

这段代码直到文档加载完才会加载指定的外部 js 文件。因此，不应该把那些页面正常加载需要依赖的 javascript 代码放在这里。而应该将 JavaScript 代码分成两组。一组是因页面需要而立即加载的 javascript 代码，另外一组是在页面加载后进行操作的 javascript 代码(例如添加 click 事件或其他东西)。这些需等到页面加载后再执行的 JavaScript 代码，应放在一个外部文件，然后再引进来

在元素中设置 defer 属性，等于告诉浏览器立即下载，但延迟执行元素中设置 defer 属性，等于告诉浏览器立即下载，但延迟执行元素中设置 defer 属性，等于告诉浏览器立即下载，但延迟执行

25、IE 与标准事件模型有哪些差别？（了解）

1. 添加事件

DOM 事件模型 – addEventListener

addEventListener(eventType, handler, useCapture)

eventType 不带有 on 字符串；

handler 参数是一个事件句柄，这个函数或方法带有一个事件对象参数；

useCapture 参数决定了事件句柄触发在何种事件传播阶段,如果 useCapture 为 true 则为捕获阶段，反之则为冒泡阶段。

IE 事件模型 – attachEvent

attachEvent(eventType, handler)

eventType 带 on 字符串;

handler 参数是一个事件句柄, 这个函数或方法带有一个事件对象参数;

2. 事件过程

DOM 事件模型包含捕获阶段和冒泡阶段, IE 事件模型只包含冒泡阶段;

DOM 事件模型可使用 e.stopPropagation()来阻止事件流

JavaScript 高级

1、typeof 和 instanceof 区别 (必会)

在 javascript 中, 判断一个变量的类型可以用 typeof

1、数字类型、typeof 返回的值是 number。比如说: typeof(1), 返回值是 number

2、字符串类型, typeof 返回的值是 string。比如 typeof("123" 返回值时 string)

3、布尔类型, typeof 返回的值是 boolean。比如 typeof(true)返回值时 boolean

4、对象、数组、null 返回的值是 object。比如 typeof(window), typeof(document),

typeof(null)返回的值都是 object

5、函数类型, 返回的值是 function。比如: typeof(eval), typeof(Date)返回的值都是 function。

6、不存在的变量、函数或者 undefined, 将返回 undefined。比如: typeof(abc)、typeof(undefined)都返回 undefined

在 javascript 中, instanceof 用于判断某个对象是否被另一个函数构造

使用 `typeof` 运算符时采用引用类型存储值会出现一个问题，无论引用的是什么类型的对象，它都返回“`object`”。ECMAScript 引入了另一个 JavaScript 运算符 `instanceof` 来解决这个问题。

`instanceof` 运算符与 `typeof` 运算符相似，用于识别正在处理的对象的类型。与 `typeof` 方法不同的是，`instanceof` 方法要求开发者明确地确认对象为某特定类型

2、js 使用 `typeof` 能得到的哪些类型？（必会）

`typeof` 只能区分值类型

`typeof undefined // undefined`

`typeof null // object`

`typeof console.log // function`

`typeof NaN // number`

3、解释一下什么是回调函数，并提供一个简单的例子？（必会）

软件模块之间总是存在着一定的接口，从调用方式上，可以把他们分为三类：同步调用、回调和异步调用

同步调用是一种阻塞式调用，调用方要等待对方执行完毕才返回，它是一种单向调用；回调是一种双向调用模式，也就是说，被调用方在接口被调用时也会调用对方的接口；

异步调用是一种类似消息或事件的机制，不过它的调用方向刚好相反，接口的服务在收到某种讯息或发生某种事件时，会主动通知客户方（即调用客户方的接口）。回调和异步调用的关系非常紧密，通常我们使用回调来实现异步消息的注册，通过异步调用来实现消息的通知。同步

调用是三者当中最简单的，而回调又常常是异步调用的基础，因此，下面我们着重讨论回调机制

在 不同软件架构中的实现

回调函数就是一个通过函数指针调用的函数。如果你把函数的指针（地址）作为参数传递给另一个函数，当这个指针被用来调用其所指向的函数时，我们就说这是回调函数。回调函数不是由该函数的实现方直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行响应

案例：

```
#include<stdio.h>
//callbackTest.c
//1.定义函数 onHeight（回调函数）
//@onHeight 函数名
//@height 参数
//@context 上下文
void onHeight(double height, void *context)
{
    printf("current height is %lf", height);
}
//2.定义 onHeight 函数的原型
//@CallbackFun 指向函数的指针类型
//@height 回调参数，当有多个参数时，可以定义一个结构体
//@context 回调上下文，在 C 中一般传入 nullptr，在 C++中可传入对象指针
typedef void (*CallbackFun)(double height, void *context);
//定义全局指针变量
CallbackFun m_pCallback;
//定义注册回调函数
void registHeightCallback(CallbackFun callback, void *context)
{
    m_pCallback = callback;
}
//定义调用函数
void printHeightFun(double height)
{
    m_pCallback(height, NULL);
}
//main 函数
```

```
int main()
{
    //注册回调函数 onHeight
    registHeightCallback(onHeight, NULL);
    //打印 height
    double h = 99;
    printHeightFun(99);
}
```

4、什么是闭包？（必会）

定义：

一个作用域可以访问另外一个函数内部的局部变量 ,或者说一个函数（子函数）访问另一个函数（父函数）中的变量。 此时就会有闭包产生 ,那么这个变量所在的函数我们就称之为闭包函数。

```
function aaa() {
    var a = 0;
    return function () {
        alert(a++);
    };
}
var fun = aaa();
fun(); //1
```

优缺点：

闭包的主要作用: 延伸了变量的作用范围, 因为闭包函数中的局部变量不会等着闭包函数执行完就销毁, 因为还有别的函数要调用它 , 只有等着所有的函数都调用完了他才会销毁
闭包会造成内存泄漏, 如何解决: 用完之后手动释放

详解：

闭包不仅仅可以实现函数内部的作用域访问这个函数中的局部变量,
还可以实现全局作用域或者是别的地方的作用域也可以访问到函数内部的局部变量 ,

实现方法就是 return 了一个函数

所以 return 函数也是我们实现闭包的一个主要原理, 因为返回的这个函数本身就是我们 fn 函数内部的一个子函数, 所以子函数是可以访问父函数里面的局部变量的, 所以返回完毕之后, 外面的函数一调用, 就会回头调用返回的这个函数, 所以就可以拿到这个子函数对应的父函数里面的局部变量.

注意:

- 1、由于闭包会使得函数中的变量都被保存在内存中, 内存消耗很大, 所以不能滥用闭包, 否则会造成网页的性能问题, 在 IE 中可能导致内存泄露。解决方法是, 在退出函数之前, 将不使用的局部变量全部删除。
- 2、闭包会在父函数外部, 改变父函数内部变量的值。所以, 如果你把父函数当作对象 (object) 使用, 把闭包当作它的公用方法 (Public Method), 把内部变量当作它的私有属性 (private value), 这时一定要小心, 不要随便改变父函数内部变量的值。

5、什么是内存泄漏 (必会)

内存泄露是指: 内存泄漏也称作"存储渗漏", 用动态存储分配函数动态开辟的空间, 在使用完毕后未释放, 结果导致一直占据该内存单元。直到程序结束。(其实说白了就是该内存空间使用完毕之后未回收)即所谓内存泄漏。

6、哪些操作会造成内存泄漏？（必会）

1、垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用数量为 0（没有其他对象引用过该对象），或对该对象的唯一引用是循环的，那么该对象的内存即可回收

2、setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏

3、闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

7、JS 内存泄漏的解决方式（必会）

1、global variables：对未声明的变量的引用在全局对象内创建一个新变量。在浏览器中，全局对象就是 window。

```
function foo(arg) {  
    bar = 'some text'; // 等同于 window.bar = 'some text';  
}
```

1.1) 解决：

1.1.1) 创建意外的全局变量

```
function foo() {  
    this.var1 = 'potential accident'  
}
```

1.1.2) 可以在 JavaScript 文件开头添加 “use strict”，使用严格模式。这样在严格模式下解析 JavaScript 可以防止意外的全局变量

1.1.3) 在使用完之后，对其赋值为 null 或者重新分配

1.2) 被忘记的 Timers 或者 callbacks

在 JavaScript 中使用 setInterval 非常常见

大多数库都会提供观察者或者其它工具来处理回调函数，在他们自己的实例变为不可达时，

会让回调函数也变为不可达的。对于 `setInterval`，下面这样的代码是非常常见的：

```
var serverData = loadData();
setInterval(function() {
  var renderer = document.getElementById('renderer');
  if(renderer) {
    renderer.innerHTML = JSON.stringify(serverData);
  }
}, 5000); //This will be executed every ~5 seconds.
```

这个例子阐述着 `timers` 可能发生的情况：计时器会引用不再需要的节点或数据

1.3) 闭包：一个可以访问外部（封闭）函数变量的内部函数

JavaScript 开发的一个关键方面就是闭包：一个可以访问外部（封闭）函数变量的内部函

数。由于 JavaScript 运行时的实现细节，可以通过以下方式泄漏内存：

```
var theThing = null;
var replaceThing = function () {
  var originalThing = theThing;
  var unused = function () {
    if (originalThing) // a reference to 'originalThing'
      console.log("hi");
  };
  theThing = {
    longStr: new Array(1000000).join('*'),
    someMethod: function () {
      console.log("message");
    }
  };
};
setInterval(replaceThing, 1000);
```

1.4) DOM 引用

有时候，在数据结构中存储 DOM 结构是有用的。假设要快速更新表中的几行内容。将每行 DOM 的引用存储在字典或数组中可能是有意义的。当这种情况发生时，就会保留同一 DOM 元素的两份引用：一个在 DOM 树中，另一个在字典中。如果将来某个时候你决定要删除这些行，则需要让两个引用都不可达。

```
var elements = {
  button: document.getElementById('button'),
```

```
        image: document.getElementById('image')
    };
    function doStuff() {
        elements.image.src = 'http://example.com/image_name.png';
    }
    function removeImage() {
        // The image is a direct child of the body element.
        document.body.removeChild(document.getElementById('image'));
        // At this point, we still have a reference to #button in the
        // global elements object. In other words, the button element is
        // still in memory and cannot be collected by the GC.
    }
}
```

8、说说你对原型 (prototype) 理解 (必会)

JavaScript 是一种通过原型实现继承的语言与别的高级语言是有区别的，像 java，C#是通过类型决定继承关系的，JavaScript 是动态的弱类型语言，总之可以认为 JavaScript 中所有都是对象，在 JavaScript 中，原型也是一个对象，通过原型可以实现对象的属性继承，JavaScript 的对象中都包含了一个“ prototype” 内部属性，这个属性所对应的就是该对象的原型

“prototype” 作为对象的内部属性，是不能被直接访问的。所以为了方便查看一个对象的原型，Firefox 和 Chrome 内核的 JavaScript 引擎中提供了“ proto ”这个非标准的访问器（ECMA 新标准中引入了标准对象原型访问器“ Object.getPrototypeOf(object)” ）

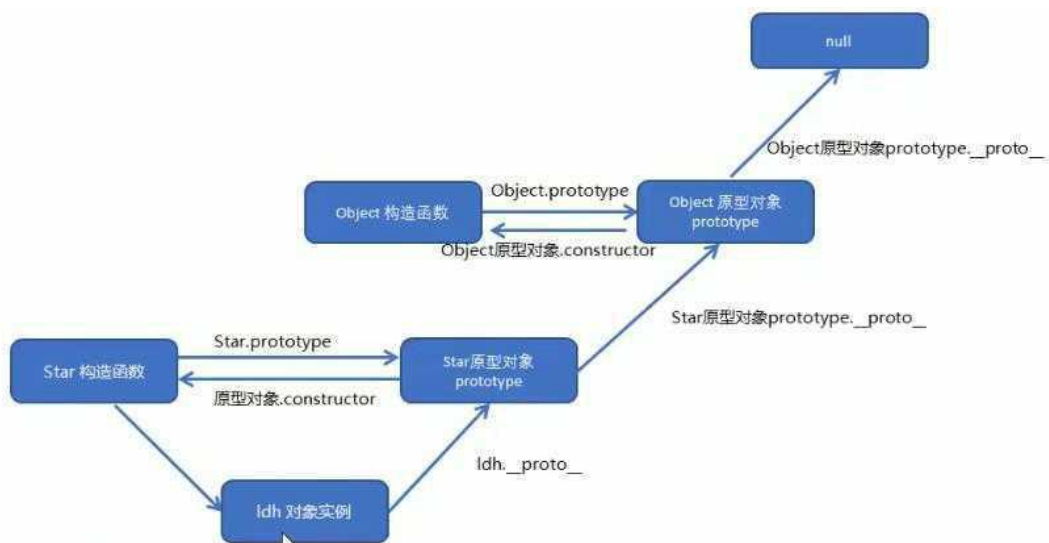
原型的主要作用就是为了实现继承与扩展对象

9、介绍下原型链（解决的是继承问题吗）（必会）

JavaScript 原型： 每个对象都会在其内部初始化一个属性，就是 prototype(原型)

原型链:

当访问一个对象的某个属性时,会先在这个对象本身属性上查找,如果没有找到,则会去它的`_proto_`隐式原型上查找,即它的构造函数的`prototype`,如果还没有找到就会再在构造函数的`prototype`的`_proto_`中查找,这样一层一层向上查找就会形成一个链式结构,我们称为原型链。



特点:

JavaScript 对象是通过引用来传递的,我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时,与之相关的对象也会继承这一改变

10、常见的 js 中的继承方法有哪些 (必会)

ES5 继承有以下六种方法:

- 1、原型链继承 JavaScript 实现继承的基本思想: 通过原型将一个引用类型继承另一个引用类型的属性和方法
- 2、借用构造函数继承(伪造对象或经典继承) JavaScript 实现继承的基本思想: 在子类构造

函数内部调用超类型构造函数。 通过使用 `apply()` 和 `call()` 方法可以在新创建的子类对象上执行构造函数

3、组合继承(原型+借用构造)(伪经典继承) JavaScript 实现继承的基本思想: 将原型链和 借用构造函数的技术组合在一块, 从而发挥两者之长的一种继承模式

将原型链和借用构造函数的技术组合到一起, 从而取长补短发挥两者长处的一种继承模式

4、型式继承 JavaScript 实现继承的基本思想: 借助原型可以基于已有的对象创建新对象, 同时还必须因此创建自定义的类型

5、寄生式继承 JavaScript 实现继承的基本思想: 创建一个仅用于封装继承过程的函数, 该函数在内部以某种方式来增强对象, 最后再像真正是它做了所有工作一样返回对象。

寄生式继承是原型式继承的加强版

6、寄生组合式继承 JavaScript 实现继承的基本思想: 通过借用函数来继承属性, 通过原型链的混成形式来继承方法

ES6 的继承:

1、使用 `class` 构造一个父类

```
class Parent {  
  constructor(name,age){  
    this.name = name  
    this.age = age  
  }  
  sayName(){  
    console.log(this.name);  
  }  
}
```

2、使用 class 构造一个子类，并使用 extends 实现继承，super 指向父类的原型对象

```
class Child extends Parent{
  constructor(name,age,gender){
    super(name,age)
    this.gender = gender
  }
  sayGender(){
    console.log(this.gender);
  }
}
```

3、实例化对象

```
const ming = new Child('ming',18,'男')
ming.sayGender()
ming.sayName()
console.log(ming.name);
console.log(ming.age);
```

11、介绍 this 各种情况（必会）

this 的情况：

- 1、以函数形式调用时，this 永远都是 window
- 2、以方法的形式调用时，this 是调用方法的对象
- 3、以构造函数的形式调用时，this 是新创建的那个对象
- 4、使用 call 和 apply 调用时，this 是指定的那个对象
- 5、箭头函数：箭头函数的 this 看外层是否有函数

如果有，外层函数的 this 就是内部箭头函数的 this

如果没有，就是 window

6、特殊情况：通常意义上 this 指针指向为最后调用它的对象。这里需要注意的一点就是如果返回值是一个对象，那么 this 指向的就是那个返回的对象，如果返回值不是一个对象那么 this 还是指向函数的实例

12、数组中的 forEach 和 map 的区别？（必会）

forEach 和 map 的相同点

相同点 都是循环遍历数组中的每一项

forEach 和 map 方法里每次执行匿名函数都支持 3 个参数，参数分别是 item（当前每一项），index（索引值），arr（原数组）

匿名函数中的 this 都是指向 window 只能遍历数组 都不会改变原数组 区别 map 方法

1.map 方法返回一个新的数组，数组中的元素为原始数组调用函数处理后的值

2.map 方法不会对空数组进行检测，map 方法不会改变原始数组。

3.浏览器支持：chrome、Safari1.5+、opera 都支持，IE9+，若 arr 为空数组，则 map 方法返回的也是一个空数组。forEach 方法

1.forEach 方法用来调用数组的每个元素，将元素传给回调函数

2.forEach 对于空数组是不会调用回调函数的。无论 arr 是不是空数组，forEach 返回的都是 undefined。这个方法只是将数组中的每一项作为 callback 的参数执行一次

13、for in 和 for of 的区别（必会）

1、推荐在循环对象属性时使用 for...in，在遍历数组的时候使用 for...of

2、for...in 循环出的是 key，for...of 循环出的是 value

3、注意，for...of 是 ES6 新引入的特性。修复了 ES5 引入的 for...in 的不足

4、for...of 不能循环普通的对象，需要通过和 Object.keys()搭配使用

14、Call 和 apply, bind 的区别（必会）

共同点：1、都是用来改变函数的 this 对象的指向的。

2、第一个参数都是 this 要指向的对象。

3、都可以利用后续参数传参。

call 方法调用一个函数，其具有一个指定的 this 值和分别地提供的参数(参数的列表)。

注意：该方法的作用和 apply() 方法类似，只有一个区别，就是 call()方法接受的是若干个参数的列表，而 apply()方法接受的是一个包含多个参数的数组

方法调用一个具有给定 this 值的函数，以及作为一个数组（或类似数组对象）提供的参数。

注意：call()方法的作用和 apply() 方法类似，区别就是 call()方法接受的是参数列表，而 apply()方法接受的是一个参数数组

bind()方法创建一个新的函数，当这个新的函数被调用时，其 this 值为提供的值，其参数列表前几项，置为创建时指定的参数序列

15、EventLoop 事件循环机制（必会）

什么是 Event Loop

JavaScript 的事件分两种，宏任务(macro-task)和微任务(micro-task)

宏任务：包括整体代码 script, setTimeout, setInterval

微任务：Promise.then(非 new Promise), process.nextTick(node 中)

事件的执行顺序——先执行宏任务，然后执行微任务，任务有同步的任务和异步的任务，同步的进入主线程，异步的进入 Event Table 并注册函数，异步事件完成后，会将回调函数放在队列中，如果还有异步的宏任务，那么就会进行循环执行上述的操作。

```
setTimeout(() => {  
  console.log('延时 1 秒');  
},1000)  
console.log("开始")  
  
//开始  
//延时 1 秒
```

上述代码，setTimeout 函数是宏任务，且是异步任务，因此会将函数放入 Event Table 并注册函数，经过指定时间后，把要执行的任务加入到 Event Queue 中，等待同步任务

console.log(“开始”)执行结束后，读取 Event Queue 中 setTimeout 的回调函数执行。

上述代码不包含微任务，接下来看包含微任务的代码：

```
setTimeout(function() {  
  console.log('setTimeout');  
},1000)  
  
new Promise(function(resolve) {  
  console.log('promise');  
}).then(function() {  
  console.log('then');  
})  
  
console.log('console');
```

- 1、首先 setTimeout，放入 Event Table 中，1 秒后将回调函数放入宏任务的 Event Queue 中
- 2、new Promise 同步代码，立即执行 console.log('promise'),然后看到微任务 then，因此将其放入微任务的 Event Queue 中
- 3、接下来执行同步代码 console.log('console')

4、主线程的宏任务，已经执行完毕，接下来要执行微任务，因此会执行 Promise.then，到此，第一轮事件循环执行完毕

5、第二轮事件循环开始，先执行宏任务，即 setTimeout 的回调函数，然后查找是否有微任务，没有，事件循环结束

总结：

事件循环先执行宏任务，其中同步任务立即执行，异步任务加载到对应的 Event Queue 中，微任务也加载到对应的微任务的 Event Queue 中，所有的同步微任务执行完之后，如果发现微任务的 Event Queue 中有未执行完的任务，先执行他们这样算是完成了一轮事件循环。接下来查看宏任务的队列中是否有异步代码，有的话执行第二轮的事件循环，以此类推。

再来看一个复杂点的例子：

```
console.log('1');
setTimeout(function() {
  console.log('2');
  process.nextTick(function() {
    console.log('3');
  })
  new Promise(function(resolve) {
    console.log('4');
    resolve();
  }).then(function() {
    console.log('5')
  })
})

//1、2、4、3、5
```

1、宏任务同步代码 console.log('1')

2、setTimeout，加入宏任务 Event Queue，没有发现微任务，第一轮事件循环走完

-
- 3、第二轮事件循环开始，先执行宏任务，从宏任务 Event Queue 中独取出 setTimeout 的回调函数
 - 4、同步代码 console.log('2'),发现 process.nextTick, 加入微任务 Event Queue
 - 5、new Promise, 同步执行 console.log('4'),发现 then, 加入微任务 Event Queue
 - 6、宏任务执行完毕, 接下来执行微任务, 先执行 process.nextTick, 然后执行 Promise.then
 - 7、微任务执行完毕, 第二轮事件循环走完, 没有发现宏任务, 事件循环结束

16、js 如何处理防抖和节流（必会）

在进行窗口的 resize、scroll，输入框内容校验等操作时，如果事件处理函数调用的频率无限制，会加重浏览器的负担，导致用户体验非常糟糕

此时我们可以采用 debounce（防抖）和 throttle（节流）的方式来减少调用频率，同时又不影响实际效果

函数防抖：

函数防抖（debounce）：当持续触发事件时，一定时间段内没有再触发事件，事件处理函数才会执行一次，如果设定的时间到来之前，又一次触发了事件，就重新开始延时

如下，持续触发 scroll 事件时，并不执行 handle 函数，当 1000 毫秒内没有触发 scroll 事件时，才会延时触发 scroll 事件

```
function debounce(fn, wait) {  
  var timeout = null;  
  return function() {  
    if(timeout !== null) clearTimeout(timeout);  
    timeout = setTimeout(fn, wait);  
  } } // 处理函数 function handle() {  
  console.log(Math.random()); }  
// 滚动事件 window.addEventListener('scroll', debounce(handle, 1000));
```

函数节流

函数节流 (throttle)：当持续触发事件时，保证一定时间段内只调用一次事件处理函数

节流通俗解释就比如我们水龙头放水，阀门一打开，水哗哗的往下流，秉着勤俭节约的优良传统美德，我们要把水龙头关小点，最好是如我们心意按照一定规律在某个时间间隔内一滴一滴的往下滴

如下，持续触发 scroll 事件时，并不立即执行 handle 函数，每隔 1000 毫秒才会执行一次

handle 函数

```
var throttle = function(func, delay) {  
    var prev = Date.now();  
    return function() {  
        var context = this;  
        var args = arguments;  
        var now = Date.now();  
        if (now - prev >= delay) {  
            func.apply(context, args);  
            prev = Date.now();  
        }  
    }  
}  
  
function handle() {console.log(Math.random());}  
window.addEventListener('scroll', throttle(handle, 1000));
```

总结：

函数防抖：将几次操作合并为一此操作进行。原理是维护一个计时器，规定在延迟时间后触发函数，但是在延迟时间内再次触发的话，就会取消之前的计时器而重新设置。这样一来，只有最后一次操作能被触发

函数节流：使得一定时间内只触发一次函数。原理是通过判断是否到达一定时间来触发函数

区别：

函数节流不管事件触发有多频繁，都会保证在规定时间内一定会执行一次真正的事件处理

函数，而函数防抖只是在最后一次事件后才触发一次函数。比如在页面的无限加载场景下，我们需要用户在滚动页面时，每隔一段时间发一次 Ajax 请求，而不是在用户停下滚动页面操作时才去请求数据。这样的场景，就适合用节流技术来实现

结合应用场景

防抖(debounce)

search 搜索联想，用户在不断输入值时，用防抖来节约请求资源。

window 触发 resize 的时候，不断的调整浏览器窗口大小会不断的触发这个事件，用防抖来让其只触发一次

节流(throttle)

鼠标不断点击触发，mousedown(单位时间内只触发一次)

监听滚动事件，比如是否滑到底部自动加载更多，用 throttle 来判断

17、New 操作符具体干了什么呢？（必会）

- 1、创建一个空对象: 并且 this 变量引入该对象,同时还继承了函数的原型
- 2、设置原型链 空对象指向构造函数的原型对象
- 3、执行函数体 修改构造函数 this 指针指向空对象,并执行函数体
- 4、判断返回值 返回对象就用该对象,没有的话就创建一个对象

18、用 JavaScript 实现冒泡排序。数据为 23、45、18、37、92、13、24（必会）

```
//升序算法
function sort(arr){
  for (var i = 0; i < arr.length; i++) {
    for (var j = 0; j < arr.length-i; j++) {
      if(arr[j]>arr[j+1]){
        var c=arr[j];//交换两个变量的位置
        arr[j]=arr[j+1];
        arr[j+1]=c;
      }
    };
  };
  return arr.toString();
}
console.log(sort([23,45,18,37,92,13,24]));
```

19、用 js 实现随机选取 10–100 之间的 10 个数字，存入一个数组并排序（必会）

```
function randomNub(aArray, len, min, max) {
  if (len >= (max - min)) {
    return '超过' + min + '-' + max + '之间的个数范围' + (max - min - 1) + '个的总数';
  }
  if (aArray.length >= len) {
    aArray.sort(function(a, b) {
      return a - b
    });
    return aArray;
  }
  var nowNub = parseInt(Math.random() * (max - min - 1)) + (min + 1);
  for (var j = 0; j < aArray.length; j++) {
    if (nowNub == aArray[j]) {
      randomNub(aArray, len, min, max);
      return;
    }
  }
}
```



```
}  
aArray.push(nowNub);  
randomNub(aArray, len, min, max);  
return aArray;  
}  
var arr=[];  
randomNub(arr,10,10,100);
```

20、已知数组 `var stringArray = ["This" , "is" , "Baidu" , "Campus"]` , Alert 出 " This is Baidu Campus" (必会)

```
var stringArray = ["This", "is", "Baidu", "Campus"]  
alert(stringArray.join(""))
```

21、已知有字符串 `foo=" get-element-by-id"` ,写一个 function 将其转化成驼峰表示法 `getElementById` (必会)

```
function combo(msg){  
var arr=msg.split("-");  
for(var i=1;i<arr.length;i++){  
arr[i]=arr[i].charAt(0).toUpperCase()+arr[i].substr(1,arr[i].length-1);  
}  
msg=arr.join("");  
return msg;  
}
```

22、有这样一个 URL:

`http://item.taobao.com/item.htm?a=1&b=2&c=&d=xxx&e` , 请写一段 JS 程序提取 URL 中的各个 GET 参数(参数名和参数个数不确定) , 将其按 key-value 形式返回到一个 json 结构中, 如 `{a: "1", b: "2", c: "", d: "xxx", e: undefined}` (必会)

```
function serilizeUrl(url) {  
var urlObject = {};
```

```
if (/^?/.test(url)) {
    var urlString = url.substring(url.indexOf("?") + 1);
    var urlArray = urlString.split("&");
    for (var i = 0, len = urlArray.length; i < len; i++) {
        var urlItem = urlArray[i];
        var item = urlItem.split("=");
        urlObject[item[0]] = item[1];
    }
    return urlObject;
}
return null;
}
```

23、输出今天的日期, 以 YYYY-MM-DD 的方式, 比如今天是 2014 年 9 月 26 日, 则输出 2014-09-26 (必会)

```
var d = new Date();
// 获取年, getFullYear()返回 4 位的数字
var year = d.getFullYear();
// 获取月, 月份比较特殊, 0 是 1 月, 11 是 12 月
var month = d.getMonth() + 1;
// 变成两位
month = month < 10 ? '0' + month : month;
// 获取日
var day = d.getDate();
day = day < 10 ? '0' + day : day;

alert(year + '-' + month + '-' + day);}
```

24、把两个数组合并, 并删除第二个元素。 (必会)

```
var array1 = ['a','b','c'];
var bArray = ['d','e','f'];
var cArray = array1.concat(bArray);
cArray.splice(1,1);
```

25、写一个 function，清除字符串前后的空格。（兼容所有浏览器） (必会)

```
//使用自带接口 trim(), 考虑兼容性:
if (!String.prototype.trim) {
  String.prototype.trim = function() {
    return this.replace(/^\s+/, "").replace(/\s+$/, "");
  }
}

// test the function
var str = " \t\n test string ".trim();
alert(str == "test string"); // alerts "true"
```

26、截取字符串 abcdefg 的 efg （必会）

```
alert('abcdefg'.substring(4));
```

27、判断一个字符串中出现次数最多的字符，统计这个次数（必会）

```
var str = 'asdfsaaasasasasaa';
var json = {};
for (var i = 0; i < str.length; i++) {
  if(!json[str.charAt(i)]){
    json[str.charAt(i)] = 1;
  }else{
    json[str.charAt(i)]++;
  }
};

var iMax = 0;
var iIndex = "";
for(var i in json){
  if(json[i]>iMax){
    iMax = json[i];
    iIndex = i;
  }
}

alert('出现次数最多的是:' + iIndex + '出现' + iMax + '次');
```

28、将数字 12345678 转化成 RMB 形式 如： 12,345,678 （必会）

```
//思路：先将数字转为字符， str= str + '' ;  
//利用反转函数，每三位字符加一个 ','最后一位不加； re()是自定义的反转函数，最后再反转回去！  
for(var i = 1; i <= re(str).length; i++){  
    tmp += re(str)[i - 1];  
    if(i % 3 == 0 && i != re(str).length){  
        tmp += ',';  
    }  
}
```

29、Split () 和 join () 的区别？（必会）

Split()是把一串字符（根据某个分隔符）分成若干个元素存放在一个数组里

即切割成数组的形式；

join() 是把数组中的字符串连成一个长串，可以大体上认为是 Split()的逆操作

30、JavaScript 中如何对一个对象进行深度 clone？（必会）

```
<!doctype html>  
<html>  
<head>  
<meta charset="utf-8">  
<title>深克隆</title>  
<script>  
function clone(obj)  
{  
    if(typeof obj== 'object')  
    {  
        if(obj instanceof Array)  
        {  
            var result=[];  
            for(var i=0;i<obj.length;i++)  
            {  
                result[i]=clone(obj[i]);  
            }  
        }  
    }  
}
```

```
}
return result;
}
else
{
var result={};
for(var i in obj)
{
result[i]=clone(obj[i]);
}
return result;
}
}
else
{
return obj;
}
}
var obj1=[12, {a: 11, b: 22}, 5];
var obj2=clone(obj1);
obj2[1].a+=5;
console.log(obj1, obj2);
</script>
</head>
<body>
</body>
</html>
```

31、js 数组去重，能用几种方法实现（必会）

- 1、使用 es6 set 方法 [...new Set(arr)] let arr = [1,2,3,4,3,2,3,4,6,7,6]; let unique = (arr)=> [...new Set(arr)]; unique(arr);//[1, 2, 3, 4, 6, 7]
- 2、利用新数组 indexOf 查找 indexOf() 方法可返回某个指定的元素在数组中首次出现的位置。如果没有就返回-1。
- 3、for 双重循环 通过判断第二层循环，去重的数组中是否含有该元素，如果有就退出第

二 层循环，如果没有 `j==result.length` 就相等，然后把对应的元素添加到最后的数组里面。

```
let arr = [1,2,3,4,3,2,3,4,6,7,6]; let result = []; for(var i = 0 ; i
< arr.length; i++) {
  for(var j = 0 ; j < result.length ; j++) {
    if( arr[i] === result[j]){
      break;
    }
  };
  if(j == result.length){
    result.push(arr[i]);
  };
};
console.log(result);
```

4、利用 for 嵌套 for，然后 splice 去重

```
function unique(arr){ for(var i=0; i<arr.length;
i++){ for(var j=i+1; j<arr.length; j++){ if(arr[i]==arr[j]){
//第一个等同于第二个，splice 方法删除第二个 arr.splice(j,1); j--; } } } return arr; }
```

5、利用 filter

```
let arr = [1,2,3,4,3,2,3,4,6,7,6]; let unique = (arr) => {
return arr.filter((item,index) => {
  return arr.indexOf(item) === index;
}) }; unique(arr);
```

6、利用 Map 数据结构去重

```
let arr = [1,2,3,4,3,2,3,4,6,7,6]; let unique = (arr)=> {
```

```
let seen = new Map();

return arr.filter((item) => {

    return !seen.has(item) && seen.set(item,1);

});

}; unique(arr);
```

32、谈谈你对 Javascript 垃圾回收机制的理解？（高薪常问）

1、标记清除（mark and sweep）

这是 JavaScript 最常见的垃圾回收方式，当变量进入执行环境的时候，比如函数中声明一个变量，垃圾回收器将其标记为“进入环境”，当变量离开环境的时候（函数执行结束）将其标记为“离开环境”

垃圾回收器会在运行的时候给存储在内存中的所有变量加上标记，然后去掉环境中的变量以及被环境中变量所引用的变量（闭包），在这些完成之后仍存在标记的就是要删除的变量了

2、引用计数(reference counting)

在低版本 IE 中经常会出现内存泄露，很多时候就是因为其采用引用计数方式进行垃圾回收。引用计数的策略是跟踪记录每个值被使用的次数，当声明了一个变量并将一个引用类型赋值给该变量的时候这个值的引用次数就加 1，如果该变量的值变成了另外一个，则这个值得引用次数减 1，当这个值的引用次数变为 0 的时候，说明没有变量在使用，这个值没法被访问了，因此可以将其占用的空间回收，这样垃圾回收器会在运行的时候清理掉引用次数为 0 的值占用的空间

在 IE 中虽然 JavaScript 对象通过标记清除的方式进行垃圾回收，但 BOM 与 DOM 对象却是通过引用计数回收垃圾的，也就是说只要涉及 BOM 及 DOM 就会出现循环引用问题

33、Class 和普通构造函数有何区别？（高薪常问）

Js 构造函数：

```
function MathHandle(x,y){
  this.x=x
  this.y=y
}
MathHandle.prototype.add=function(){
  Return this.x+this.y
}
var m =new MathHandle(1,2)
console.log(m.add())
class 基本语法：
class MathHandle{
  constructor(x,y){
    this.x = x
    this.y = y
  }
  add(){
    return this.x + this.y
  }
}
const m = new MathHandle(1,2)
console.log(m.add())
```

语法糖：

在上述两段代码中分别加入如下代码，运行

```
console.log(typeof MathHandle) // 'function'
```

```
console.log(MathHandle.prototype.constructor === MathHandle) //true
```

```
console.log(m.__proto__ === MathHandle.prototype) //true
```

运行结果一致。我认为，class 是构造函数的语法糖

综上所述：

Class 在语法上更加贴合面向对象的写法

Class 实现继承更加易读、易理解

更易于写 java 等后端语言的使用

本质还是语法糖，使用 prototype

34、JS 里垃圾回收机制是什么，常用的是哪种，怎么处理的？（高薪常问）

JS 的垃圾回收机制是为了以防内存泄漏，内存泄漏的含义就是当已经不需要某块内存时这块内存还存在着，垃圾回收机制就是间歇的不定期的寻找到不再使用的变量，并释放掉它们所指向的内存

JS 中最常见的垃圾回收方式是标记清除

工作原理：是当变量进入环境时，将这个变量标记为“进入环境”。当变量离开环境时，则将其标记为“离开环境”。标记“离开环境”的就回收内存

工作流程：

垃圾回收器，在运行的时候会给存储在内存中的所有变量都加上标记

去掉环境中的变量以及被环境中的变量引用的变量的标记

再被加上标记的会被视为准备删除的变量

垃圾回收器完成内存清除工作，销毁那些带标记的值并回收他们所占用的内存空间

35、什么是进程、什么是线程、它们之间是什么关系（了解）

1、进程：

1.1) 程序执行时的一个实例

1.2) 每个进程都有独立的内存地址空间

1.3) 系统进行资源分配和调度的基本单位

1.4) 进程里的堆，是一个进程中最大的一块内存，被进程中的所有线程共享的，进程创建时分配，主要存放 new 创建的对象实例

1.5) 进程里的方法区，是用来存放进程中的代码片段的，是线程共享的

1.6) 在多线程 OS 中，进程不是一个可执行的实体，即一个进程至少创建一个线程

去执行代码

2、线程：

2.1) 进程中的一个实体

2.2) 进程的一个执行路径

2.3) CPU 调度和分派的基本单位

2.4) 线程本身是不会独立存在

2.5) 当前线程 CPU 时间片用完后，会让出 CPU 等下次轮到自己时候在执行

2.6) 系统不会为线程分配内存，线程组之间只能共享所属进程的资源

2.7) 线程只拥有在运行中必不可少的资源(如程序计数器、栈)

2.8) 线程里的程序计数器就是为了记录该线程让出 CPU 时候的执行地址，待再次分

配到时间片时候就可以从自己私有的计数器指定地址继续执行

2.9) 每个线程有自己的栈资源，用于存储该线程的局部变量和调用栈帧，其它线程

无权访问

3、关系：

3.1) 一个程序至少一个进程，一个进程至少一个线程，进程中的多个线程是共享进

程的资源

3.2) Java 中当我们启动 main 函数时就启动了一个 JVM 的进程，而 main 函数所在线程就是这个进程中的一个线程，也叫做主线程

3.3) 一个进程中有多个线程，多个线程共享进程的堆和方法区资源，但是每个线程

有自己的程序计数器，栈区域

36、什么是任务队列？（了解）

任务队列（task queue）主要分两种：

1、宏任务（macrotask）：在新标准中叫 task

1.1) 主要包括：script(整体代码)，setTimeout，setInterval，setImmediate，I/O，
ui rendering

2、微任务（microtask）：在新标准中叫 jobs

2.1) 主要包括：process.nextTick，Promise，MutationObserver (html5 新特性)

3、扩展：

3.1) 同步任务：在主线程上，排队执行的任务，只有前一个任务执行完毕，才能执

行后一个任务

3.2) 异步任务：不进入主线程，而进入“任务队列”的任务，只有“任务队列”

通知主线程，某个异步任务可以执行了，该任务才会进入主线程执行

37、栈和队列的区别？（了解）

- 1、栈的插入和删除操作都是在一端进行的，而队列的操作却是在两端进行的
- 2、队列先进先出，栈先进后出
- 3、栈只允许在表尾一端进行插入和删除，而队列只允许在表尾一端进行插入，在表头一端进行删除#

38、栈和堆的区别？（了解）

- 1、栈区 (stack) — 由编译器自动分配释放，存放函数的参数值，局部变量的值等。堆区 (heap) — 一般由程序员分配释放，若程序员不释放，程序结束时可能由 OS 回收
- 2、堆（数据结构）：堆可以被看成是一棵树，如：堆排序；栈（数据结构）：一种先进后出的数据结构

jQuery

1、jQuery 的 `$(document).ready(function () {})`, `$(function () {})` 与原生 JS 的 `window.onload` 有什么不同？（必会）

1.执行时间

`window.onload` 必须等到页面内包括图片、音频、视频在内的所有元素加载完毕后才能执行
`$(document).ready()` 是 DOM 结构绘制完毕后就执行，而无需对图像或外部资源加载的等待，

从而执行起来更快

2.编写个数不同

window.onload 不能同时编写多个，如果有多个 window.onload 方法，只会执行一个

\$(document).ready()可以同时编写多个，并且都可以得到执行

3.简化写法

window.onload 没有简化写法

\$(document).ready(function(){}))可以简写成\$(function(){}))

2、jQuery 和 Zepto 的区别？各自的使用场景？（必会）

1、同：1) Zepto 最初是为移动端开发的库，是 jQuery 的轻量级替代品，因为它的 API 和 jQuery 相似，而文件更小

2) Zepto 最大的优势是它的文件大小，只有 8k 多，是目前功能完备的库中最小的一个，尽管不大，Zepto 所提供的工具足以满足开发程序的需要

3)大多数在 jQuery 中常用的 API 和方法 Zepto 都有，Zepto 中还有一些 jQuery 中没有的。

4) 因为 Zepto 的 API 大部分都能和 jQuery 兼容，所以用起来极其容易，如果熟悉 jQuery，就能很容易掌握 Zepto。你可用同样的方式重用 jQuery 中的很多方法，也可以方便的把方法串在一起得到更简洁的代码，甚至不用看它的文档。

2、异：1) Zepto 更轻量级

2) Zepto 是 jQuery 的精简，针对移动端去除了大量 jQuery 的兼容代码

3) 针对移动端程序，Zepto 有一些基本的触摸事件可以用来做触摸屏交互（tap 事件、swipe 事件），Zepto 是不支持 IE 浏览器的。

4) DOM 操作的区别：添加 id 时 jQuery 不会生效而 Zepto 会生效

5) 事件触发的区别：使用 jQuery 时 load 事件的处理函数不会执行；使用 zepto 时 load 事件的处理函数会执行

6) 事件委托的区别: zepto 中, 选择器上所有的委托事件都依次放入到一个队列中, 而在 jQuery 中则委托成独立的多个事件

7) width() 与 height()的区别: zepto 由盒模型 (box-sizing) 决定, 用.width()返回赋值的 width, 用.css('width')返回 border 等的结果; jQuery 会忽略盒模型, 始终返回内容区域的宽/高 (不包含 padding、border)

8) offset()的区别: zepto 返回{top,left,width,height}; jQuery 返回{width,height}。zepto 无法获取隐藏元素宽高, jQuery 可以

9) zepto 中没有为原型定义 extend 方法而 jQuery 有

10) zepto 的 each 方法只能遍历数组, 不能遍历 JSON 对象

3、你是如何使用 jQuery 中的 ajax 的? (必会)

1、\$.ajax, 这个是 jQuery 对 ajax 封装的最基础函数, 通过使用这个函数可以完成异步通讯的所有功能。也就是说什么情况下我们都可以通过此方法进行异步刷新的操作。但是它的参数较多, 有的时候可能会麻烦一些。看一下常用的参数:

```
var configObj = {  
    method //数据的提交方式: get 和 post  
    url //数据的提交路劲  
    async //是否支持异步刷新, 默认是 true  
    data //需要提交的数据  
    dataType //服务器返回数据的类型, 例如 xml,String,Json 等  
    success //请求成功后的回调函数  
    error //请求失败后的回调函数  
}  
  
$.ajax(configObj);//通过$.ajax 函数进行调用。
```

2、\$.post, 这个函数其实就是对\$.ajax 进行了更进一步的封装, 减少了参数, 简化了操作, 但是运用的范围更小了。\$.post 简化了数据提交方式, 只能采用 POST 方式提交。只能是异步访问服务器, 不能同步访问, 不能进行错误处理。在满足这些情况下, 我们可以使用这个函数来方便我们的编程, 它的主要几个参数, 像 method, async 等进行了默认设置, 我们不可以改变的。

url:发送请求地址。
data:待发送 Key/value 参数。
callback:发送成功时回调函数。
type:返回内容格式, xml, html, script, json, text_default。

3、\$.get, 和\$.post 一样, 这个函数是对 get 方法的提交数据进行封装, 只能使用在 get 提交数据解决异步刷新的方式上, 使用方式和\$.post 差不多。

四, \$.getJSON, 这个是进一步的封装, 也就是对返回数据类型为 Json 进行操作。里边就三个参数, 需要我们设置, 非常简单: url,[data],[callback]。

4、jQuery 的常用的方法增、删、复制、改、查 (必会)

1、插入

append(content) : 将 content 内容插入到匹配元素内容的最后

prepend(content) : 将 content 内容插入到匹配元素内容的最前

2、删除

empty()将内容清空标签还在

remove()指定的标签和内容都移除

3、复制

`clone([true])`

参数说明：有 `true`：克隆元素和元素绑定的事件，没有 `true`：只克隆元素

4、替换

`replaceWith()`

5、查找

`eq(index)`：查找指定下标的元素下标从 0 开始

`filter(expr)`：过滤匹配的 class 选择器，其实就是缩小范围查找

`not(expr)`：排除匹配指定选择器之外的元素

`next([expr])`：查找指定元素下一个元素

`prev([expr])`：查找指定元素的上一个元素

`parent([expr])`：查找当前元素的父元素

5、jQuery 中 `$.get()` 提交和 `$.post()` 提交的区别？（必会）

相同点：都是异步请求的方式来获取服务端的数据；

异同点：1、请求方式不同：`$.get()` 方法使用 GET 方法来进行异步请求的。`$.post()` 方法使用 POST 方法来进行异步请求的。

2、参数传递方式不同：`get` 请求会将参数跟在 URL 后进行传递，而 `POST` 请求则是作为 HTTP

消息的实体内容发送给 Web 服务器的，这种传递是对用户不可见的。

3、数据传输大小不同：get 方式传输的数据大小不能超过 2KB 而 POST 要大的多

4、安全问题：GET 方式请求的数据会被浏览器缓存起来，因此有安全问题。

6、简单的讲叙一下 jQuery 是怎么处理事件的，你用过哪些事件？ (必会)

首先去加载文档，在页面加载完毕后，浏览器会通过 javascript 为 DOM 元素添加事件

jQuery 中的常用事件

.click()鼠标单击触发 du 事件

.dblclick()双击触发

.mousedown()/up()鼠标按下/弹起触发事件

.mousemove(), 鼠标移动事件; .mouseover()/out(), 鼠标移入/移出触发事件

.mouseenter()/leave()鼠标进入/离开触发事件

.hover(func1,func2), 鼠标移入调用 func1 函数，移出调用 func2 函数

.focusin(), 鼠标聚焦到该元素时触发事件

.focusout(), 鼠标失去焦点时触发事件

.focus()/blur()鼠标聚焦/失去焦点触发事件（不支持冒泡）

.change(), 表单元素发生改变时触发事件

.select(), 文本元素被选中时触发事件

.submit(), 表单提交动作触发

.keydown()/up(), 键盘按键按下/弹起触发

.on(), 多事件的绑定

7、你使用过 jQuery 中的动画吗，是怎样用的？（必会）

使用过。

- 1) hide()和 show()同时修改多个样式属性，像高度，宽度，不透明度；
- 2) fadeIn()和 fadeOut()fadeTo()只改变不透明度
- 3) slideUp()和 slideDown()slideToggle()只改变高度；
- 4) animate()属于自定义动画的方法。

8、你在 jQuery 中使用过哪些插入节点的方法，它们的区别是什么？（必会）

append(),appendTo(),prepend(),prependTo(),after(),insertAfter(), before(),insertBefore()

大致可以分为内部追加和外部追加

append()表示向每个元素内部追加内容

appendTo()将所有匹配的元素追加到指定的元素中

prepend(): 向每个匹配的元素内部前置添加内容

prependTo(): 将所有匹配的元素前置到指定的元素中

after(): 在每个匹配元素之后插入内容

insertAfter(): 将所有配的元素插入到指定元素的后面

例\$(A).appendTo(B)是将 A 追加到 B 中下面的方法解释类似

9、jQuery 中如何来获取或和设置属性？（必会）

jQuery 中可以用 `attr()`方法来获取和设置元素属性，`removeAttr()`方法来删除元素属性

10、jQuery 如何设置和获取 HTML、文本和值？（必会）

1、`html()`方法：如果想更改或者是设置 HTML 的内容，我们可以使用 `html()`方法，首先我们先使用这个方法获取元素里面的内容 `var html=$("p").html()`。如果需要设置某元素的 HTML 代码，那么我们就可以使用此方法加上一个参数。此方法只能应用于 XHTML 中，不能用于 xml。

2、`text()`方法，去设置某个元素中的文本内容，代码是 `var text=$("p").text()`；如果想设置文本同样需要给它传一个参数。

3、`val()`方法，可以用来设置和获取元素的值，它不仅仅可以设置元素，同时也能获取元素，另外，它能是下拉列表框，多选框，和单选框相应的选项被选中，在表单操作中会经常用到。

11、有哪些查询节点的选择器？（必会）

`:first` 查询第一个

`:last` 查询最后一个

`:odd` 查询奇数但是索引从 0 开始

`:even` 查询偶数

`:eq(index)`查询相等的

`:gt(index)`查询大于 index 的

`:lt` 查询小于 index:header 选取所有的标题等

12、jQuery 中的 hover()和 toggle()有什么区别？（必会）

1、hover()和 toggle()都是 jQuery 中两个合成事件

hover(fn1,fn2): 一个模仿悬停事件（鼠标移动到一个对象上面及移出这个对象）的方法。当鼠

2、标移动到一个匹配的元素上面时，会触发指定的第一个函数。当鼠标移出这个元素时，会触发指定的第二个函数

3、toggle(evenFn,oddFn):每次点击时切换要调用的函数。如果点击了一个匹配的元素，则触发指定的第一个函数，当再次点击同一元素时，则触发指定的第二个函数。随后的每次点击都重复对这两个函数的轮番调用

13、jQuery 中 detach()和 remove()方法的区别是什么？（必会）

detach()和 remove()作用相同，即移除被选元素，包括所有文本和子节点

不同之处在于 detach():移除被选元素，包括所有文本和子节点。会保留所有绑定的事件、附加的数据

remove():移除被选元素，包括所有文本和子节点。绑定的事件、附加的数据等都会被移除

14、\$(this)和 this 关键字在 jQuery 中有何不同？（必会）

\$(this)返回一个 jQuery 对象，你可以对它调用多个 jQuery 方法，比如用 text()获取文本，用 val()获取值等等。

而 this 代表当前元素，它是 JavaScript 关键词中的一个，表示上下文中的当前 DOM 元素。你不能对它调用 jQuery 方法，直到它被 \$()函数包裹，例如\$(this)。

15、jQuery 中 attr()和 prop()的区别（必会）

1、对于 HTML 元素本身就带有的固有属性，或者说 W3C 标准里就包含有这些属性，更直观的说法就是，编辑器里面可以智能提示出来的一些属性，如：src、href、value、class、name、id 等。在处理时，使用 prop()方法。

2、对于 HTML 元素我们自定义的 DOM 属性，即元素本身是没有这个属性的，如：data-*。在处理时，使用 attr()方法。

```
<a href="#" id="link1" class="btn" action="delete">删除</a>
```

这个例子中的 <a> 元素的 dom 属性值有 "id、href、class 和 action"，很明显，前三个是固有属性，而后面一个 action 属性是我们自己定义上去的

<a> 元素本身是没有属性的。这种就是自定义的 dom 属性。处理这些属性时，建议使用 attr 方法，使用 prop 方法对自定义属性取值和设置属性值时，都会返回 undefined 值。

像 checkbox、radio 和 select 这样的元素，选中属性对应 "checked" 和 "selected"，这些也属于固有属性，因此需要使用 prop 方法去操作才能获取正确答案

16、jQuery 库中的 \$() 是什么？（必会）

\$() 函数是 jQuery() 函数的别称，\$() 函数用于将任何对象包裹成 jQuery 对象，然后被允许调用定义在 jQuery 对象上的多个不同方法。甚至可以将一个选择器字符串传入 \$() 函数，它会返回一个包含所有匹配的 DOM 元素数组的 jQuery 对象。

17、jQuery 的属性拷贝(extend)的实现原理是什么，如何实现深浅拷贝？（高薪常问）

jQuery.extend() 函数用于将一个或多个对象的内容合并到目标对象。

语法

```
$.extend( target [, object1 ] [, objectN ] )
```

指示是否深度合并

```
$.extend( [deep ], target, object1 [, objectN ] )
```

注意:不支持第一个参数传递 false 。

参数	描述
deep	可选。 Boolean 类型 指示是否深度合并对象，默认为 false。 如果该值为 true，且多个对象的某个同名属性也都是对象， 则该"属性对象"的属性也将进行合并。
target	Object 类型 目标对象，其他对象的成员属性将被附加到该对象上。
object1	可选。 Object 类型 第一个被合并的对象。
objectN	可选。 Object 类型 第 N 个被合并的对象。

深拷贝，深拷贝代码把 extend 函数的第一个参数设置为 true：

（对原始对象属性所引用的对象进行递归拷贝）`var newObject = $.extend(true, {},oldObject);`

浅拷贝，浅拷贝代码 extend 函数里不传入第一个参数，默认为 false（只复制一份原始对象的引用）
`var newObject = $.extend({}, oldObject);`

18、jQuery 的实现原理？（高薪常问）

- 1、为了防止全局变量污染，把 jQuery 的代码写在一个自调函数中
- 2、咱们平常使用的 \$ 实际上 jQuery 对外暴露的一个工厂函数
- 3、而构造函数在 jQuery 的内部叫 init，并且这个构造函数还被添加到了 jQuery 的原型中。
当我们调用工厂函数的时候返回的其实是一个构造函数的实例
- 4、jQuery 为了让第三方能够对其功能进行扩展，所以把工厂函数的原型与构造函数的原型保持了一致。这样子对外暴露工厂函数，即可对原型进行扩展

数据可视化

1、echarts 的基本用法（必会）

1、初始化类

Html 里面创建一个 id 为 box1 的 div，并初始化 echarts 绘图实例
`var myChart = echarts.init(document.getElementById('box1'))`

2、样式配置

- title：标题
- tooltip：鼠标悬停气泡
- xAxis：配置横轴类别，type 类型为 category 类别

-
- series: 销量数据, data 参数与横轴一一对应, 如果想调样式, 也可以简单调整, 比如
每个条形图的颜色可以通过函数进行数组返回渲染

3、渲染图展示表

```
myChart.setOption(option);
```

2、如何使用 echarts (必会)

①获取 echarts :在官网下载 echarts 版本 或 npm 下载 ②引入 echarts :script 引入 或者 vue 在入口文件里引用 ③创建一个 dom 元素 用来放置图表 ④配置 echarts 属性

3、echarts 如何画图? (必会)

1、echarts 是通过 canvas 来实现的, 由于 canvas 的限制, 所以 echarts 在实现的时候多是绘制一些规则的, 可预期的, 易于实现的东西

2、echarts 的核心就是 options 配置的对象。一般使用最多的是直角坐标图, 极点图, 饼状图, 地图。

3、对于直角坐标, 必须配置 xAxis 和 yAxis, 对于极点坐标必须配置 radiusAxis 和 angleAxis。

4、就是 series 系列的认识, 它是一个数组, 数组的每一项都代表着一个单独的系列, 可以配置各种图形等功能。然后 data

一般是一个每一项都是数组的数组, 也就是嵌套数组。里层数组一般代表坐标位置

4、echarts 绘制条形图 (必会)

1、初始化类

Html 里面创建一个 id 为 box1 的 div，并初始化 echarts 绘图实例

```
var myChart = echarts.init(document.getElementById('box1'))
```

2、样式配置

title ： 标题

tooltip ： 鼠标悬停气泡

xAxis：配置横轴类别，type 类型为 category 类别

series: 销量数据，data 参数与横轴一一对应，如果想调样式，也可以简单调整，比如每个条形图的颜色可以通过函数进行数组返回渲染

3、渲染图展示表

```
myChart.setOption(option);
```

5、切换其他组件统计图时，出现卡顿问题如何解决（必会）

1、原因：每一个图例在没有数据的时候它会创建一个定时器去渲染气泡，页面切换后，echarts 图例是销毁了，但是这个 echarts 的实例还在内存当中，同时它的气泡渲染定时器还在运行。这就导致 echarts 占用 CPU 高，导致浏览器卡顿，当数据量比较大时甚至浏览器崩溃

2、解决方法：在 mounted()方法和 destroy()方法之间加一个 beforeDestroy()方法释放该页面的 chart 资源，clear()方法则是清空图例数据，不影响图例的 resize，而且能够释放内存，切换的时候就很顺畅了

```
beforeDestroy () {  
  this.chart.clear()  
}
```

```
}
```

6、echarts 图表自适应 div resize 问题（必会）

echarts 官网的实例都具有响应式功能

echarts 图表本身是提供了一个 resize 的函数的。

用于当 div 发生 resize 事件的时候，让其触发 echarts 的 resize 事件，重绘 canvas。

```
<div class="chart">
  <div class="col-md-3" style="width:73%;height:270px" id="chartx"></div>
</div>
<script src="/static/assets/scripts/jquery.ba-resize.js"></script>
js 代码:
var myChartx = echarts.init(document.getElementById('chartx'));
$('.chart').resize(function(){
    myChartx.resize();
})
```

注：jQuery 有 resize()事件，但直接调用没有起作用，引入 jquery.ba-resize.js 文件

7、echarts 在 vue 中怎么引用？（必会）

首先我们初始化一个 vue 项目，执行 vue init webpack echart,接着我们进入初始化的项目下。

安装 echarts,

```
npm install echarts -S //或
cnpm install echarts -S
```

安装完成之后，我们就可以开始引入我们需要的 echarts 了，接下来介绍几种使用 echarts 的方式。

全局引用：

首先在 main.js 中引入 echarts，将其绑定到 vue 原型上：

```
import echarts from 'echarts'
Vue.prototype.$echarts = echarts;
```

接着，我们就可以在任何一个组件中使用 echarts 了。

局部使用：

当然，很多时候没必要在全局引入 echarts，那么我们只在单个组件内使用即可，代码更加简单：

```
<template>
  <div>
    <div style="width:500px;height:500px" ref="chart"> </div>
  </div>
</template>
<script>
const echarts = require('echarts');
export default{
  data () {
    return {};
  },
  methods: {
    initCharts () {
      let myChart = echarts.init(this.$refs.chart);
      // 绘制图表
      myChart.setOption({
        title: { text: '在 Vue 中使用 echarts' },
        tooltip: {},
        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]
        }]
      });
    }
  }
}
```

```
},  
mounted () {  
  this.initCharts();  
}  
}  
</script>
```

可以看到，我们直接在组件内引入 echarts，接下来跟全局引入的使用一样。区别在于，这种方式如果你想在其他组件内用 echarts，则必须重新引入了。

8、echarts 支持哪些图标？（了解）

折线图（区域图）、柱状图（条状图）、散点图（气泡图）、K 线图、饼图（环形图）

雷达图（填充雷达图）、和弦图、力导向布局图、地图、仪表盘、漏斗图、事件流程图等 12 类图表

Ajax/计算机网络相关

1、什么是 Ajax, Ajax 的原理, Ajax 都有哪些优点和缺点？（必会）

什么是 Ajax

Ajax 是 “Asynchronous JavaScript and XML” 的缩写。他是指一种创建交互式网页应用的网页 开发技术。沟通客户端与服务器，可以在不必刷新整个浏览器的情况下，与服务器进行异步通讯的技术

Ajax 的原理

通过 XMLHttpRequest 对象来向服务器发异步请求,从服务器获得数据,然后用 javascript 来 操作 DOM 而更新页面。这其中最关键的一步就是从服务器获得请求数据。

XMLHttpRequest 是 Ajax 的核心机制,它是在 IE5 中首先引入的,是一种支持异步请求的技术。简单的说,也就是 javascript 可以及时向服务器提出请求和处理响应,而不阻塞用户。达到无刷新的效果。

Ajax 的优点

- 1、最大的一点是页面无刷新,用户的体验非常好。
- 2、使用异步方式与服务器通信,具有更加迅速的响应能力。
- 3、可以把以前一些服务器负担的工作转嫁到客户端,利用客户端闲置的能力来处理,减轻服务器和带宽的负担,节约空间和宽带租用成本。并且减轻服务器的负担, Ajax 的原则是“按需取数据”,可以最大程度的减少冗余请求,和响应对服务器造成的负担。
- 4、基于标准化的并被广泛支持的技术,不需要下载插件或者小程序。

Ajax 的缺点

- 1、Ajax 不支持浏览器 back 按钮。
- 2、安全问题 Ajax 暴露了与服务器交互的细节。
- 3、对搜索引擎的支持比较弱。
- 4、破坏了程序的异常机制。
- 5、不容易调试。

2、常见的 HTTP 状态码以及代表的意义 (必会)

5 种常见的 HTTP 状态码以及代表的意义

200 (OK) : 请求已成功, 请求所希望的响应头或数据体将随此响应返回。

303 (See Other) : 告知客户端使用另一个 URL 来获取资源。

400 (Bad Request) : 请求格式错误。1)语义有误, 当前请求无法被服务器理解。除非进行修改, 否则客户端不应该重复提交这个请求; 2)请求参数有误。

404 (Not Found) : 请求失败, 请求所希望得到的资源未被在服务器上发现。

500 (Internal Server Error) : 服务器遇到了一个未曾预料的状态, 导致了它无法完成对请求的处理。

更多状态码

100 => 正在初始化 (一般是看不到的)

101 => 正在切换协议 (websocket 浏览器提供的)

202 => 表示接受

301 => 永久重定向/永久转移

302 => 临时重定向/临时转移 (一般用来做服务器负载均衡)

304 => 本次获取的内容是读取缓存中的数据, 会每次去服务器校验

401 => 未认证, 没有登录网站

403 => 禁止访问, 没有权限

502 => 充当网关或代理的服务器, 从远端服务器接收到了一个无效的请求

503 => 服务器超负荷 (假设一台服务器只能承受 10000 人, 当第 10001 人访问的时候, 如果服务器没有做负载均衡, 那么这个人的网络状态码就是 503)

505 => 服务器不支持请求的 HTTP 协议的版本, 无法完成处理。

3、请介绍一下 XMLHttpRequest 对象及常用方法和属性（必会）

XMLHttpRequest 对象

Ajax 的核心是 JavaScript 对象 XMLHttpRequest。该对象在 Internet Explorer 5 中首次引入，它是一种支持异步请求的技术。简而言之，XMLHttpRequest 使您可以使用 JavaScript 向服务器提出请求并处理响应，而不阻塞用户。通过 XMLHttpRequest 对象，Web 开发人员可以在页面加载以后进行页面的局部更新

方法

`open(String method,String url,boolean asynch,String username,String password)`

`send(content)`

`setRequestHeader(String header,String value)`

`getAllResponseHeaders()`

`getResponseHeader(String header)`

`abort()`

常用详细解析

`open()`：该方法创建 HTTP 请求

第一个参数是指定提交方式(post、get)

第二个参数是指定要提交的地址是哪

第三个参数是指定是异步还是同步(true 表示异步，false 表示同步)

第四和第五参数在 HTTP 认证的时候会用到。是可选的

`setRequestHeader(String header,String value)`: 设置消息头 (使用 post 方式才会使用到, `get` 方法并不需要调用该方法)

`xmlHTTP.setRequestHeader("Content-type","application/x-www-form-urlencoded");`

`send(content)`: 发送请求给服务器

如果是 `get` 方式, 并不需要填写参数, 或填写 `null`

如果是 `post` 方式, 把要提交的参数写上去

常用属性

`onreadystatechange`: 请求状态改变的事件触发器 (`readyState` 变化时会调用此方法), 一般用于指定回调函数

`readyState`: 请求状态 `readyState` 一改变, 回调函数被调用, 它有 5 个状态

0: 未初始化

1: `open` 方法成功调用以后

2: 服务器已经应答客户端的请求

3: 交互中。HTTP 头信息已经接收, 响应数据尚未接收。

4: 完成。数据接收完成

`responseText`: 服务器返回的文本内容

`responseXML`: 服务器返回的兼容 DOM 的 XML 内容

`status`: 服务器返回的状态码

`statusText`: 服务器返回状态码的文本信息

回调函数是什么

回调函数就是接收服务器返回的内容!

4、Ajax 的实现流程是怎样的? (必会)

- 1、创建 XMLHttpRequest 对象,也就是创建一个异步调用对象.
- 2、创建一个新的 HTTP 请求,并指定该 HTTP 请求的方法、URL 及验证信息.
- 3、设置响应 HTTP 请求状态变化的函数.
- 4、发送 HTTP 请求.
- 5、获取异步调用返回的数据.
- 6、使用 JavaScript 和 DOM 实现局部刷新.

```
<script type="text/javascript">
```

```
var HTTPRequest;
```

```
function checkUsername() {
```

```
    //创建 XMLHttpRequest 对象
```

```
    if(window.XMLHttpRequest) {
```

```
        //在 IE6 以上的版本以及其他内核的浏览器(Mozilla)等
```

```
        HTTPRequest = new XMLHttpRequest();
```

```
    }else if(window.ActiveXObject) {
```

```
        //在 IE6 以下的版本
```

```
        HTTPRequest = new ActiveXObject();
```

```
    }
```

```
    //创建 HTTP 请求
```

```
HttpRequest.open("POST", "Servlet1", true);
```

```
//因为我使用的是 post 方式，所以需要设置消息头
```

```
HttpRequest.setRequestHeader("Content-type",  
    "application/x-www-form-urlencoded");
```

```
//指定回调函数
```

```
HttpRequest.onreadystatechange = response22;
```

```
//得到文本框的数据
```

```
var name = document.getElementById("username").value;
```

```
//发送 HTTP 请求，把要检测的用户名传递进去
```

```
HttpRequest.send("username=" + name);
```

```
}
```

```
//接收服务器响应数据
```

```
function response22() {
```

```
    //判断请求状态码是否是 4【数据接收完成】
```

```
    if(HttpRequest.readyState==4) {
```

```
        //再判断状态码是否为 200【200 是成功的】
```

```
        if(HttpRequest.status==200) {
```

```
            //得到服务端返回的文本数据
```

```
            var text = HttpRequest.responseText;
```

```
            //把服务端返回的数据写在 div 上
```

```
            var div = document.getElementById("result");
```

```
div.innerText = text;
```

```
}}</script>
```

5、Ajax 接收到的数据类型有哪些，数据如何处理？（必会）

接收到的数据类型

String /JSON 字符串/JSON 对象

JSON 对象直接循环使用

JSON 字符串转 JSON 使用

String 直接使用

如何处理数据

1、字符串转对象

第一种方式：eval () ；

```
var data='{ "student": [{"name": "张三", "age": "11"}, {"name": "李四", "age": "11"}, {"name": "王 五", "age": "11"}] }' ;  
  
eval ( ' ( "+data+" ) ' ) ;
```

第二种方式：JSON.parse () ；

```
var data='{ "student": [{"name": "张三", "age": "11"}, {"name": "李四", "age": "11"}, {"name": "王 五", "age": "11"}] }' ;  
  
JSON.parse (data) ;
```

parse () 与 eval () 区别

eval () 方法不会去检查给的字符串时候符合 json 的格式~同时如果给的字符串中存在 js

代 码 eval () 也会一并执行~比如:

```
var data='{ "student": [ {"name": "张三", "age": "11"}, {"name": "李四", "age": "alert(11)"}, {"name": "王五", "age": "11"} ] }' ;
```

此时执行 eval 方法后会先弹出一个提示框输出 11 的字符串;

这时候使用 JSON.parse()就会报错,显示错误信息为当前字符串不符合 json 格式;即

JSON.parse()方法会检查需要转换的字符串是否符合 json 格式

相比而言 eval () 方法是很不安全, 特别是当涉及到第三方时我们需要确保传给 eval ()

的 参数是我们可以控制的, 不然里面插入比如 window.location~指向一个恶意的连接总的来

说, 还是推荐使用 JSON.parse () 来实现 json 格式字符串的解析

2、对象转字符串

JSON.stringify(json)

6、请解释一下 JavaScript 的同源策略 (必会)

同源策略是客户端脚本 (尤其是 Javascript) 的重要的安全度量标准。它最早出自 Netscape

Navigator2.0, 其目的是防止某个文档或脚本从多个不同源装载。所谓同源指的是: 协议,

域名, 端口相同, 同源策略是一种安全协议, 指一段脚本只能读取来自同一起来源的窗口和

文档的属性。

7、为什么会有跨域的问题出现, 如何解决跨域问题 (必会)

什么是跨域

指的是浏览器不能执行其他网站的脚本，它是由浏览器的同源策略造成的,是浏览器对

javascript 施加的安全限制，防止他人恶意攻击网站

比如一个黑客,他利用 iframe 把真正的银行登录页面嵌到他的页面上,当你使用真实的用户名和密码登录时,如果没有同源限制,他的页面就可以通过 JavaScript 读取到你的表单中输入的内容,这样用户名和密码就轻松到手了。

解决方式

1、jsonp

原理：动态创建一个 script 标签。利用 script 标签的 src 属性不受同源策略限制。因为所有的 src 属性和 href 属性都不受同源策略限制。可以请求第三方服务器数据内容。

步骤

1.1) 去创建一个 script 标签

1.2) script 的 src 属性设置接口地址

1.3) 接口参数,必须要带一个自定义函数名 要不然后台无法返回数据。

1.4) 通过定义函数名去接收后台返回数据

```
//去创建一个 script 标签
```

```
var script = document.createElement("script");
```

```
//script 的 src 属性设置接口地址 并带一个 callback 回调函数名称
```

```
script.src = "HTTP://127.0.0.1:8888/index.php?callback=jsonpCallback";
```

```
//插入到页面
```

```
document.head.appendChild(script);
```

```
//通过定义函数名去接收后台返回数据 function jsonpCallback(data){
```

```
//注意 jsonp 返回的数据是 json 对象可以直接使用
```

```
//Ajax 取得数据是 json 字符串需要转换成 json 对象才可以使用。
```

```
}
```

2、CORS：跨域资源共享

原理：服务器设置 Access-Control-Allow-Origin HTTP 响应头之后，浏览器将会允许跨域

请 求

限制：浏览器需要支持 HTML5，可以支持 POST，PUT 等方法兼容 ie9 以上

需要后台设置

Access-Control-Allow-Origin: * //允许所有域名访问，或者

Access-Control-Allow-Origin: HTTP://a.com //只允许所有域名访问

3、反向代理

4、window+iframe

8、Get 和 Post 的区别以及使用场景（必会）

区别

1、Get 使用 URL 或 Cookie 传参。而 Post 将数据放在 BODY 中

2、Get 的 URL 会有长度上的限制，则 Post 的数据则可以非常大

3、Post 比 Get 安全，因为数据在地址栏上不可见

最本质的区别

Get 是用来从服务器上获得数据，而 post 是用来向服务器上传递数据

Get/Post 使用场景

若符合下列任一情况，则 Post 方法：

- 1、请求的结果有持续性的作用，例如：数据库内添加新的数据行
- 2、若使用 Get 方法，则表单上收集的数据可能让 URL 过长
- 3、要传送的数据不是采用 ASCII 编码

若符合下列任一情况，则用 Get 方法：

- 1、请求是为了查找资源，html 表单数据仅用来搜索
- 2、请求结果无持续性的副作用
- 3、收集的数据及 html 表单内的输入字段名称的总长不超过 1024 个字符

9、解释 jsonp 的原理（必会）

什么是 jsonp，jsonp 的作用

jsonp 并不是一种数据格式，而 json 是一种数据格式，jsonp 是用来解决跨域获取数据的一种解决方案

具体原理

是通过动态创建 script 标签，然后通过标签的 src 属性获取 js 文件中的 js 脚本，该脚本的内容是一个函数调用，参数就是服务器返回的数据，为了处理这些返回的数据，需要事先在页面定义好回调函数，本质上使用的并不是 Ajax 技术，Ajax 请求受同源策略的影响，不允许进行跨域请求，而 script 标签的 src 属性中的链接却可以访问跨域的 js 脚本，利用这个特性，服务端不在返回 json 格式的数据，而是返回调用某个函数的 js 代码，在 src 中进行了调用，这样就实现了跨域

10、封装好的 Ajax 里的常见参数及其代表的含义（必会）

url: 发送请求的地址。

type: 请求方式 (post 或 get) 默认为 get。

async: 同步异步请求, 默认 true 所有请求均为异步请求。

timeout: 超时时间设置, 单位毫秒

data: 要求为 Object 或 String 类型的参数, 发送到服务器的数据

cache: 默认为 true (当 dataType 为 script 时, 默认为 false), 设置为 false 将不会从浏览器缓存中加载请求信息。

dataType: 预期服务器返回的数据类型。

可用的类型如下:

xml: 返回 XML 文档, 可用 JQuery 处理。

html: 返回纯文本 HTML 信息; 包含的 script 标签会在插入 DOM 时执行。

script: 返回纯文本 JavaScript 代码。不会自动缓存结果。

json: 返回 JSON 数据。

jsonp: JSONP 格式。使用 JSONP 形式调用函数时, 例如 myurl?callback=?, JQuery 将自动替换后一个 “?” 为正确的函数名, 以执行回调函数。

text: 返回纯文本字符串。

success: 请求成功后调用的回调函数, 有两个参数。

- 1、由服务器返回, 并根据 dataType 参数进行处理后的数据。

- 2、描述状态的字符串。

error: 要求为 Function 类型的参数, 请求失败时被调用的函数。该函数有 3 个参数

1、XMLHttpRequest 对象

2、错误信息

3、捕获的错误对象(可选)

complete :function(XMLHttpRequest,status){ //请求完成后最终执行参数}

11、jQuery 中 Ajax 与 fetch 、 axios 有什么区别? (必会)

1、jQuery Ajax

```
$.Ajax({  
  type: 'POST',  
  url: url,  
  data: data,  
  dataType: dataType,  
  success: function () {},  
  error: function () {}  
});
```

jQuery 本身是针对 MVC 的编程,不符合现在前端 MVVM 的开发模式

jQuery 整个项目很大, 单纯使用 Ajax 却要引入整个 jQuery 非常的不合理 (采取个性化打

包的方案又不能享受 CDN 服务)

2、axios

```
axios({  
  method: 'post',  
  url: '/user/12345',  
  data: {  
    firstName: 'Fred',
```

```
        lastName: 'Flintstone'
      }) .then(function (response) {
        console.log(response);
      })
    })
```

客户端支持防止 CSRF/XSRF 自动转换 JSON 数据 取消请求 转换请求和响应数据 拦截请

求 和响应支持 Promise API 从 node.js 发出 HTTP 请求 从浏览器中创建

XMLHttpRequest axios 是一个基于 Promise 用于浏览器和 nodejs 的 HTTP 客户端

3、fetch

```
let data =response.json();

let response = await fetch(url);

try {

catch(e) { console.log(data);

console.log("Oops,error", e); }
```

为什么要用 axios

3.1) fetch 没有办法原生监测请求的进度，而 XHR 可以

3.2) fetch 不支持 abort，不支持超时控制，使用 setTimeout 及 Promise.reject 的实现 超时控制并不能阻止请求过程继续在后台运行，造成了量的浪费

3.3) fetch 默认不会带 cookie，需要添加配置项

3.5) fetch 只对网络请求报错，对 400，500 都当做成功的请求，需要封装去处理

脱离了 XHR，是 ES 规范里新的实现方式 更加底层，提供的 API 丰富（request，

response) 更好更方便的写法符合关注分离，没有将输入、输出和用事件来跟踪的状态混杂在一个对象里

12、Ajax 注意事项及适用和不适用场景（必会）

Ajax 开发时，网络延迟——即用户发出请求到服务器发出响应之间的间隔——需要慎重考虑。不给予用户明确的回应，没有恰当的预读数据，或者对 XMLHttpRequest 的不恰当处理，都会使用户感到延迟，这是用户不希望看到的，也是他们无法理解的。通常的解决方案是，使用一个可视化的组件来告诉用户系统正在进行后台操作并且正在读取数据和内容。

Ajax 适用场景

- 1、表单驱动的交互
- 2、深层次的树的导航
- 3、快速的用户与用户间的交流响应
- 4、类似投票、yes/no 等无关痛痒的场景
- 5、对数据进行过滤和操纵相关数据的场景
- 6、普通的文本输入提示和自动完成的场景

Ajax 不适用场景

- 1、部分简单的表单
- 2、搜索
- 3、基本的导航
- 4、替换大量的文本
- 5、对呈现的操纵

13、HTTP 与 HTTPS 的区别（必会）

1、HTTPS 协议需要到 CA（Certificate Authority，证书颁发机构）申请证书，一般免费证书较少，因而需要一定费用。（以前网易官网是 HTTP，而网易邮箱是 HTTPS。）

2、HTTP 是超文本传输协议，信息是明文传输，HTTPS 则是具有安全性的 SSL 加密传输协议

3、HTTP 和 HTTPS 使用的是完全不同的连接方式，用的端口也不一样，前者是 80，后者是 443

4、HTTP 的连接很简单，是无状态的。HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 HTTP 协议安全。（无状态的意思是其数据包的发送、传输和接收都是相互独立的。无连接的意思是指通信双方都不长久的维持对方的任何信息。）

14、LocalStorage、sessionStorage、cookie 的区别（必会）

共同点：都是保存在浏览器端、且同源的

区别：

1、cookie 数据始终在同源的 http 请求中携带（即使不需要），即 cookie 在浏览器和服务端间来回传递，而 sessionStorage 和 localStorage 不会自动把数据发送给服务器，仅在本地保存。cookie 数据还有路径（path）的概念，可以限制 cookie 只属于某个路径下

2、存储大小限制也不同，cookie 数据不能超过 4K，同时因为每次 http 请求都会携带 cookie、所以 cookie 只适合保存很小的数据，如会话标识。sessionStorage 和 localStorage 虽然也有存储大小的限制，但比 cookie 大得多，可以达到 5M 或更大

3、数据有效期不同，sessionStorage：仅在当前浏览器窗口关闭之前有效；localStorage：

始终有效, 窗口或浏览器关闭也一直保存, 因此用作持久数据; cookie: 只在设置的 cookie

过期时间之前有效, 即使窗口关闭或浏览器关闭

4、作用域不同, sessionStorage 不在不同的浏览器窗口中共享, 即使是同一个页面;

localStorage 在所有同源窗口中都是共享的; cookie 也是在所有同源窗口中都是共享的

5、web Storage 支持事件通知机制, 可以将数据更新的通知发送给监听者

6、web Storage 的 api 接口使用更方便

15、简述 web 前端 Cookie 机制, 并结合该机制说明会话保持原理? (必会)

Cookie 是进行网站用户身份, 实现服务端 Session 会话持久化的一种非常好方式。Cookie 最早由 Netscape 公司开发, 现在由 IETF 的 RFC 6265 标准对其规范, 已被所有主流浏览器所支持

1、为什么需要 Cookie

HTTP 是一种无状态的协议, 客户端与服务器建立连接并传输数据, 数据传输完成后, 连接就会关闭。再次交互数据需要建立新的连接, 因此, 服务器无法从连接上跟踪会话, 也无法知道用户上一次做了什么。这严重阻碍了基于 Web 应用程序的交互, 也影响用户的交互体验。如: 在网络有时候需要用户登录才进一步操作, 用户输入用户名密码登录后, 浏览了几个页面, 由于 HTTP 的无状态性, 服务器并不知道用户有没有登录

Cookie 是解决 HTTP 无状态性的有效手段, 服务器可以设置或读取 Cookie 中所包含的信息。

当用户登录后, 服务器会发送包含登录凭据的 Cookie 到用户浏览器客户端, 而浏览器对该

Cookie 进行某种形式的存储（内存或硬盘）。用户再次访问该网站时，浏览器会发送该 Cookie（Cookie 未到期时）到服务器，服务器对该凭据进行验证，合法时使用户不必输入用户名和密码就可以直接登录

本质上讲，Cookie 是一段文本信息。客户端请求服务器时，如果服务器需要记录用户状态，就在响应用户请求时发送一段 Cookie 信息。客户端浏览器保存该 Cookie 信息，当用户再次访问该网站时，浏览器会把 Cookie 做为请求信息的一部分提交给服务器。服务器检查 Cookie 内容，以此来判断用户状态，服务器还会对 Cookie 信息进行维护，必要时会对 Cookie 内容进行修改

2、Cookie 的类型

Cookie 总时由用户客户端进行保存的（一般是浏览器），按其存储位置可分为：内存式 Cookie 和硬盘式 Cookie。

内存式 Cookie 存储在内存中，浏览器关闭后就会消失，由于其存储时间较短，因此也被称为非持久 Cookie 或会话 Cookie。

硬盘式 Cookie 保存在硬盘中，其不会随浏览器的关闭而消失，除非用户手工清理或到了过期时间。由于硬盘式 Cookie 存储时间是长期的，因此也被称为持久 Cookie。

3、Cookie 的实现原理

Cookie 定义了一些 HTTP 请求头和 HTTP 响应头，通过这些 HTTP 头信息使服务器可以与客户端进行状态交互。

客户端请求服务器后，如果服务器需要记录用户状态，服务器会在响应信息中包含一个

Set-Cookie 的响应头，客户端会根据这个响应头存储 Cookie 信息。再次请求服务器时，

客户端会在请求信息中包含一个 Cookie 请求头，而服务器会根据这个请求头进行用户身份、状态等校验。

下面是一个实现 Cookie 机制的，简单的 HTTP 请求过程：

3.1) 客户端请求服务器

客户端请求 IT 笔录网站首页，请求头如下：

GET / HTTP/1.0

HOST: itbilu.com

3.2) 服务器响应请求

Cookie 是一种 key=value 形式的字符串，服务器需要记录这个客户端请求的状态，

因此在响应头中包含一个 Set-Cookie 字段。响应头如下：

HTTP/1.0 200 OK

Set-Cookie: UserID=itbilu; Max-Age=3600; Version=1

Content-type: text/html

.....

3.3) 再次请求时，客户端请求中会包含一个 Cookie 请求头

客户端会对服务器响应的 Set-Cookie 头信息进行存储。再次请求时，将会在请求头中

包含服务器响应的 Cookie 信息。请求头如下

GET / HTTP/1.0

HOST: itbilu.com

Cookie: UserID=itbilu

16、一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么（高薪常问）

- 1、浏览器查找域名对应的 IP 地址(DNS 查询: 浏览器缓存->系统缓存->路由器缓存->ISP
DNS 缓存->根域名服务器)
- 2、浏览器向 Web 服务器发送一个 HTTP 请求 (TCP 三次握手)
- 3、服务器 301 重定向 (从 HTTP://example.com 重定向到 HTTP://www.example.com)
- 4、浏览器跟踪重定向地址，请求另一个带 www 的网址
- 5、服务器处理请求 (通过路由读取资源)
- 6、服务器返回一个 HTTP 响应 (报头中把 Content-type 设置为 'text/html')
- 7、浏览器进 DOM 树构建
- 8、浏览器发送请求获取嵌在 HTML 中的资源 (如图片、音频、视频、CSS、JS 等)
- 9、浏览器显示完成页面
- 10、浏览器发送异步请求

17、你知道的 HTTP 请求方式有几种（高薪常问）

HTTPRequestMethod 共计 17 种

- 1、GET 请求指定的页面信息，并返回实体主体。
- 2、HEAD 类似于 get 请求，只不过返回的响应中没有具体的内容，用于获取报头
- 3、POST 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和/或已有资源的修改。
- 4、PUT 从客户端向服务器传送的数据取代指定的文档的内容。

-
- 5、DELETE 请求服务器删除指定的页面。
 - 6、CONNECT HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。
 - 7、OPTIONS 允许客户端查看服务器的性能。
 - 8、TRACE 回显服务器收到的请求，主要用于测试或诊断。
 - 9、PATCH 实体中包含一个表，表中说明与该 URI 所表示的原内容的区别。
 - 10、MOVE 请求服务器将指定的页面移至另一个网络地址。
 - 11、COPY 请求服务器将指定的页面拷贝至另一个网络地址。
 - 12、LINK 请求服务器建立链接关系。
 - 13、UNLINK 断开链接关系。
 - 14、WRAPPED 允许客户端发送经过封装的请求。
 - 15、LOCK 允许用户锁定资源，比如可以再编辑某个资源时将其锁定，以防别人同时对其进行编辑。
 - 16、MKCOL 允许用户创建资源
 - 17、Extension-mothed 在不改动协议的前提下，可增加另外的方法。

18、什么是 TCP 连接的三次握手（高薪常问）

TCP 是因特网中的传输层协议，使用三次握手协议建立连接，完成三次握手，客户端与服务器开始传送数据。

第一次握手：建立连接时，客户端发送 syn 包（syn=j）到服务器，并进入 SYN_SENT 状态，等待服务器确认；SYN：同步序列编号（Synchronize Sequence Numbers）。

第二次握手：服务器收到 syn 包，必须确认客户的 SYN ($ack=j+1$)，同时自己也发送一个 SYN 包 ($syn=k$)，即 SYN+ACK 包，此时服务器进入 SYN_RECV 状态；

第三次握手：客户端收到服务器的 SYN+ACK 包，向服务器发送确认包 ACK($ack=k+1$)，此包发送完毕，客户端和服务器进入 ESTABLISHED (TCP 连接成功) 状态，完成三次握手。



TCP 连接建立图

TCP 协议优点

TCP 发送的包有序号，对方收到包后要给一个反馈，如果超过一定时间还没收到反馈就自动执行超时重发，因此 TCP 最大的优点是可靠。

TCP 协议缺点

很简单，就是麻烦，如果数据量比较小的话建立连接的过程反而占了大头，不断地重发也会造成网络延迟，因此比如视频聊天通常就使用 UDP，因为丢失一些包也没关系，速度流畅才是重要的。

19、为什么 TCP 连接需要三次握手四次挥手（高薪常问）

为什么是三次握手

为了防止已失效的连接请求报文段突然有送到了服务器，因而产生错误,假设两次握手时，客户发出的第一个请求连接报文段在某一网络节点长时间滞留，以致延误到连接释放后才到达服务器。服务器收到失效的连接请求报文段后，认为是客户又发出一次新的连接请求。于是向客户发送确认报文段，同意建立连接，此时在假定两次握手的前提下，连接建立成功。这样会导致服务器的资源白白浪费

为什么是四次挥手

TCP 协议是全双工通信，这意味着客户端和服务端都可以向彼此发送数据，所以关闭连接是双方都需要确认的共同行为，假设是三次挥手时，首先释放了客户到服务器方向的连接，此时 TCP 连接处于半关闭状态，这时客户不能向服务器发送数据，而服务器还是可以向客户发送数据。如果此时客户收到了服务器的确认报文段后，就立即发送一个确认报文段，这会导致服务器向客户还在发送数据时连接就被关闭。这样会导致客户没有完整收到服务器所发的报文段

20、TCP 与 UDP 的区别有哪些（高薪常问）

什么是 TCP

TCP (Transmission Control Protocol 传输控制协议) 是一种面向连接的、可靠的、基于字节流的传输层通信协议

什么是 UDP

UDP(User Datagram Protocol 用户数据报协议) 是 OSI (Open System Interconnection, 开放式 系统互联) 参考模型中一种无连接的传输层协议, 提供面向事务的简单不可靠信息传送服务

区别

TCP 是面向连接的传输控制协议, 而 UDP 提供了无链接的数据报服务//类似电话与短信

TCP 面向连接, 提供可靠的数据服务

TCP 首部开销 20 字节,UDP 首部开销 8 字节

TCP 逻辑通信信道是全双工的可靠信道, UDP 则是不可靠信道

UDP 没有拥塞机制, 因此网络出现拥堵不会使源主机的发送效率降低 (有利于实时会议视频 等)

TCP 的连接只能是点到点的,UDP 支持一对一, 多对一, 多对多的交互通信

21、介绍一下 websocket (高薪常问)

什么是 websocket

websocket 是一种网络通信协议, 是 HTML5 开始提供的一种在单个 TCP 连接上进行全双工通信的协议, 这个对比着 HTTP 协议来说, HTTP 协议是一种无状态的、无连接的、单向的应用层协议, 通信请求只能由客户端发起, 服务端对请求做出应答处理。HTTP 协议无法实现服务器主动向客户端发起消息, websocket 连接允许客户端和服务器之间进行全双工通信, 以便任一方都可以通过建立的连接将数据推送到另一端。websocket 只需要建立一次连接, 就可以一直保持连接状态

```
<script>
// 初始化一个 WebSocket 对象
var ws = new WebSocket("ws://localhost:9998/echo");
// 建立 web socket 连接成功触发事件
ws.onopen = function () {
// 使用 send() 方法发送数据
ws.send("发送数据");
alert("数据发送中..."); };
// 接收服务端数据时触发事件
ws.onmessage = function (evt) { var received_msg = evt.data; alert("数据已接收..."); };
// 断开 web socket 连接成功触发事件
ws.onclose = function () { alert("连接已关闭..."); };

</script>
```

22、拆解一下 URL 的各个部分，分别是什么意思（高薪常问）

例如：scheme://host:port/path?query#fragment

- 1、.scheme:通信协议，常用的 HTTP,ftp,maito 等
- 2、.host:主机，服务器(计算机)域名系统 (DNS) 主机名或 IP 地址
- 3、.port:端口号，整数，可选，省略时使用方案的默认端口，如 HTTP 的默认端口为 80
- 4、.path:路径，由零或多个"/"符号隔开的字符串，一般用来表示主机上的一个目录或文件地址
- 5、.query:查询，可选，用于给动态网页传递参数，可有多个参数，用"&"符号隔开，每个参数的名和值用"="符号隔开
- 6、.fragment:信息片断，字符串，用于指定网络资源中的片断。例如一个网页中有多个名词解释，可使用 fragment 直接定位到某一名词解释。(也称为锚点)

23、HTTP 缓存机制（高薪常问）

答案没有整理出来

ES6

1、ES5 和 ES6 的区别，说几个 ES6 的新增方法（必会）

ES5 和 ES6 的区别

ECMAScript5，即 ES5，是 ECMAScript 的第五次修订，于 2009 年完成标准化

ECMAScript6，即 ES6，是 ECMAScript 的第六次修订，于 2015 年完成，也称 ES2015

ES6 是继 ES5 之后的一次改进，相对于 ES5 更加简洁，提高了开发效率

ES6 的新增方法

1、新增声明命令 let 和 const

在 ES6 中通常用 let 和 const 来声明，let 表示变量、const 表示常量

1.1) 特点

let 和 const 都是块级作用域。以 {} 代码块作为作用域范围 只能在代码块里面使用

不存在变量提升，只能先声明再使用，否则会报错。在代码块内，在声明变量之前，

该变量 都是不可用的。这在语法上，称为“暂时性死区”（temporal dead zone，

简称 TDZ）， 在同一个代码块内，不允许重复声明

const 声明的是一个只读常量，在声明时就需要赋值。（如果 const 的是一个对象，

对 象所包含的值是可以被修改的。抽象一点儿说，就是对象所指向的地址不能改变，而

变量成员 是可以修改的。）

2、模板字符串 (Template String)

用一对反引号(`)标识，它可以当作普通字符串使用，也可以用来定义多行字符串，也可以在字符串中嵌入变量，js 表达式或函数，变量、js 表达式或函数需要写在`\${}`中。

3、函数的扩展

3.1) 函数的默认参数

ES6 为参数提供了默认值。在定义函数时便初始化了这个参数，以便在参数没有被传递进去时使用。

3.2) 箭头函数

在 ES6 中，提供了一种简洁的函数写法，我们称作“箭头函数”。

3.2.1) 写法

函数名=(形参)=>{.....} 当函数体中只有一个表达式时，{}和 return 可以省略当函数体中形参只有一个时，()可以省略。

3.2.2) 特点

箭头函数中的 this 始终指向箭头函数定义时的离 this 最近的一个函数，如果没有最近的函数就指向 window。

4、对象的扩展

4.1) 属性的简写

ES6 允许在对象之中，直接写变量。这时，属性名为变量名，属性值为变量 的值。

```
var foo = 'bar';
```

```
var baz = {foo}; //等同于 var baz = {foo: foo};
```

```
方法的简写。省略冒号与 function 关键字。
```

```
var o = {  
  method() {  
    return "Hello!";  
  }  
};
```

// 等同于

```
var o = {  
  method: function() {  
    return "Hello!";  
  }  
};
```

4.2) Object.keys()方法

获取对象的所有属性名或方法名（不包括原形的内容），返回一个数组。

```
var obj={name: "john", age: "21", getName: function () { alert(this.name)}};  
  
console.log(Object.keys(obj));    // ["name", "age", "getName"]  
  
console.log(Object.keys(obj).length);    //3  
  
console.log(Object.keys(["aa", "bb", "cc"]));    //["0", "1", "2"]  
  
console.log(Object.keys("abcdef"));    //["0", "1", "2", "3", "4", "5"]
```

4.3) Object.assign ()

assign 方法将多个原对象的属性和方法都合并到了目标对象上面。可以接收多个参数，

第一个参数是目标对象，后面的都是源对象

```
var target = {}; //目标对象
```

```
var source1 = {name : 'ming', age: '19'}; //源对象 1
```

```
var source2 = {sex : '女'}; //源对象 2
```

```
var source3 = {sex : '男'}; //源对象 3, 和 source2 中的对象有同名属性 sex
```

```
Object.assign(target,source1,source2,source3);
```

```
console.log(target);    //{name : 'ming', age: '19', sex: '男'}
```

5、for...of 循环

```
var arr=["小林","小吴","小佳"];
```

```
for(var v of arr){
```

```
    console.log(v);
```

```
}
```

```
//小林 //小吴 //小佳
```

6、import 和 export

ES6 标准中, JavaScript 原生支持模块(module)。这种将 JS 代码分割成不同功能的小块进行 模块化, 将不同功能的代码分别写在不同文件中, 各模块只需导出公共接口部分, 然后通过模块的导入的方式可以在其他地方使用

export 用于对外输出本模块（一个文件可以理解为一个模块）变量的接口

import 用于在一个模块中加载另一个含有 export 接口的模块

import 和 export 命令只能在模块的顶部, 不能在代码块之中

7、Promise 对象

Promise 是异步编程的一种解决方案，将异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数，要是为了解决异步处理回调地狱(也就是循环嵌套的问题)而产生的

Promise 构造函数包含一个参数和一个带有 resolve（解析）和 reject（拒绝）两个参数的回调。在回调中执行一些操作（例如异步），如果一切都正常，则调用 resolve，否则调用 reject。对于已经实例化过的 Promise 对象可以调用 Promise.then() 方法，传递 resolve 和 reject 方法作为回调。then()方法接收两个参数：onResolve 和 onReject，分别代表当前 Promise 对象在成功或失败时

Promise 的 3 种状态

Fulfilled 为成功的状态，Rejected 为失败的状态，Pending 既不是 Fulfilled 也不是

Rejected 的状态，可以理解为 Promise 对象实例创建时候的初始状态

8、解构赋值

8.1) 数组的解构赋值

解构赋值是对赋值运算符的扩展。

是一种针对数组或者对象进行模式匹配，然后对其中的变量进行赋值。

在代码书写上简洁且易读，语义更加清晰明了；也方便了复杂对象中数据字段获取。

数组中的值会自动被解析到对应接收该值的变量中，数组的解构赋值要一一对应如果有对应不上的就是 undefined

```
let [a, b, c] = [1, 2, 3];  
  
// a = 1 // b = 2 // c = 3
```

8.2) 对象的解构赋值

对象的解构赋值和数组的解构赋值其实类似，但是数组的数组成员是有序的

而对象的属性则是无序的，所以对象的解构赋值简单理解是等号的左边和右边的结构

相同

```
let { foo, bar } = { foo: 'aaa', bar: 'bbb' }; // foo = 'aaa' // bar = 'bbb'
```

```
let { baz : foo } = { baz : 'ddd' }; // foo = 'ddd'
```

9、Set 数据结构

Set 数据结构，类似数组。所有的数据都是唯一的，没有重复的值。它本身是一个构造函数。

9.1) Set 属性和方法

Size () 数据的长度

Add () 添加某个值，返回 Set 结构本身。

Delete () 删除某个值，返回一个布尔值，表示删除是否成功。

Has () 查找某条数据，返回一个布尔值。

Clear () 清除所有成员，没有返回值。

9.2) 主要应用场景：数组去重

10、class

class 类的继承 ES6 中不再像 ES5 一样使用原型链实现继承，而是引入 Class 这个概念

ES6 所写的类相比于 ES5 的优点：

区别于函数，更加专业化（类似于 JAVA 中的类）

写法更加简便，更容易实现类的继承

11、...

展开运算符可以将数组或对象里面的值展开；还可以将多个值收集为一个变量

12、async、await

使用 async/await, 搭配 Promise, 可以通过编写形似同步的代码来处理异步流程, 提高代码的简洁性和可读性 async 用于申明一个 function 是异步的, 而 await 用于等待一个异步方法执行完成

13、修饰器

@decorator 是一个函数, 用来修改类甚至是方法的行为。修饰器本质就是编译时执行的函数

14、Symbol

Symbol 是一种基本类型。Symbol 通过调用 symbol 函数产生, 它接收一个可选的名字参数, 该函数返回的 symbol 是唯一的

15、Proxy

Proxy 代理使用代理 (Proxy) 监听对象的操作, 然后可以做一些相应事情

2、ES6 的继承和 ES5 的继承有什么区别 (必会)

ES6 的继承和 ES5 的继承的区别

ES5 的继承是通过原型或者是构造函数机制来实现

ES6 用过 class 关键字定义类, 里面有构造方法, 类之间通过 extends 关键字实现, 子类必须在 constructor 方法中调用 super 方法

3、var、let、const 之间的区别 (必会)

区别

var 声明变量可以重复声明，而 let 不可以重复声明

var 是不受限于块级的，而 let 是受限于块级

var 会与 window 相映射（会挂一个属性），而 let 不与 window 相映射

var 可以在声明的上面访问变量，而 let 有暂存死区，在声明的上面访问变量会报错

const 声明之后必须赋值，否则会报错

const 定义不可变的量，改变了就会报错

const 和 let 一样不会与 window 相映射、支持块级作用域、在声明的上面访问变量会报错

4、Class、extends 是什么，有什么作用（必会）

什么是 Class，Class 的作用

ES6 的 Class 可以看作只是一个 ES5 生成实例对象的构造函数的语法糖。

它参考了 java 语言，定义了一个类的概念，让对象原型写法更加清晰，对象实例化更像是一种面向对象编程。

什么是 extends，extends 的作用

extends 是 ES6 引入的关键字，其本质仍然是构造函数+原型链的组合式继承。

Class 类可以通过 extends 实现继承。

Class 和 ES5 构造函数的不同点

- 1、类的内部定义的所有方法，都是不可枚举的。
- 2、ES6 的 class 类必须用 new 命令操作，而 ES5 的构造函数不用 new 也可以执行。
- 3、ES6 的 class 类不存在变量提升，必须先定义 class 之后才能实例化，不像 ES5 中可以将 构造函数写在实例化之后。

4、ES5 的继承，实质是先创造子类的实例对象 this，然后再将父类的方法添加到 this 上面。ES6 的继承机制完全不同，实质是先将父类实例对象的属性和方法，加到 this 上面（所以必须先调用 super 方法），然后再用子类的构造函数修改 this。

5、module、export、import 有什么作用（必会）

module、export、import 是 ES6 用来统一前端模块化方案的设计思路和实现方案。

export、import 的出现统一了前端模块化的实现方案，整合规范了浏览器/服务端的模块化方法，用来取代传统的 AMD/CMD、requireJS、seaJS、commonJS 等等一系列前端模块不同的实现方案，使前端模块化更加统一规范，JS 也能更加能实现大型的应用程序开发。

import 引入的模块是静态加载（编译阶段加载）而不是动态加载（运行时加载）。

import 引入 export 导出的接口值是动态绑定关系，即通过该接口，可以取到模块内部实时的值。

6、使用箭头函数应注意什么/箭头函数和普通函数的区别（必会）

区别

用了箭头函数，this 就不是指向 window，而是父级（指向是可变的）

不能够使用 arguments 对象

不能用作构造函数，这就是说不能够使用 new 命令，否则会抛出一个错误

不可以使用 yield 命令，因此箭头函数不能用作 Generator 函数

7、ES6 的模板字符串有哪些新特性？并实现一个类模板字符串的功能（必会）

模板字符串新特性

基本的字符串格式化。将表达式嵌入字符串中进行拼接。用\${}来界定

在 ES5 时我们通过反斜杠()来做多行字符串或者字符串一行行拼接。ES6 反引号(`)就能解决

类模板字符串的功能

实现一个类模板字符串的功能

```
let name = 'sunny';

let age = 21;

let str = `你好, ${name} 已经 ${age}岁了`

str = str.replace(/\${([^\}]*)}\}/g,function(){

    return eval(arguments[1]);

})

console.log(str);//你好, sunny 已经 21 岁了
```

8、介绍下 Set、Map 的区别（必会）

区别

应用场景 Set 用于数据重组，Map 用于数据储存

Set:

成员不能重复

只有键值没有键名，类似数组

可以遍历，方法有 add, delete, has

Map:

本质上是键值对的集合，类似集合

可以遍历，可以跟各种数据格式转换

9、setTimeout、Promise、Async/Await 的区别（必会）

事件循环中分为宏任务队列和微任务队列

宏任务（macrotask）：在新标准中叫 task

主要包括: script(整体代码), setTimeout, setInterval, setImmediate, I/O, ui rendering

微任务（microtask）：在新标准中叫 jobs

主要包括: process.nextTick, Promise, MutationObserver (html5 新特性)

setTimeout、Promise、Async/Await 的区别

setTimeout 的回调函数放到宏任务队列里，等到执行栈清空以后执行

Promise.then 里的回调函数会放到相应宏任务的微任务队列里，等宏任务里面的同步代码

执 行完再执行

async 函数表示函数里面可能会有异步方法，await 后面跟一个表达式

async 方法执行时，遇到 await 会立即执行表达式，然后把表达式后面的代码放到微任务

队 列里，让出执行栈让同步代码先执行

10、Promise 有几种状态，什么时候会进入 catch? (必会)

Promise 有几种状态

三个状态: pending、fulfilled、reject

两个过程: pending -> fulfilled、pending -> rejected

Promise 什么时候会进入 catch

当 pending 为 rejected 时，会进入 catch

11、ES6 怎么写 Class , 为何会出现 Class (必会)

什么是 Class, Class 的作用

ES6 的 class 可以看作是一个语法糖，它的绝大部分功能 ES5 都可以做到，新的 class 写法

只是让对象原型的写法更加清晰、更像面向对象编程的语法

ES6 怎么写 Class

```
//定义类
```

```
class Point {
```

```
  constructor(x,y) {
```

```
    //构造方法
```

```
    this.x = x; //this 关键字代表实例对象
```

```
    this.y = y;
```

```
  } toString() {
```

```
    return '(' + this.x + ',' + this.y + ')';
```

```
  }
```

```
}
```

12、Promise 只有成功和失败 2 个状态，怎么让一个函数无论成功还是失败都能被调用？（必会）

使用 Promise.all()

Promise.all () 用于将多个 Promise 实例，包装成一个新的 Promise 实例

Promise.all () 接受一个数组作为参数，数组里的元素都是 Promise 对象的实例，如果不是，就会先调用下面讲到的 Promise.resolve ()，将参数转为 Promise 实例，再进一步处理。

(Promise.all () 方法的参数可以不是数组，但必须具有 Iterator 接口，且返回的每个成员都是 Promise 实例。)

示例：var p =Promise.all([p1,p2,p3])

p 的状态由 p1、p2、p3 决定，分为两种情况。

当该数组里的所有 Promise 实例都进入 Fulfilled 状态：Promise.all**返回的实例才会变成 Fulfilled 状态。并将 Promise 实例数组的所有返回值组成一个数组，传递给 Promise.all 返回实例的回调函数**。

当该数组里的某个 Promise 实例都进入 Rejected 状态：Promise.all 返回的实例会立即变成 Rejected 状态。并将第一个 rejected 的实例返回值传递给 Promise.all 返回实例的回调函数

13、ES6 如何转化为 ES5，为什么要转化（必会）

ES6 语法为什么要转化 ES5 语法

ECMAScript2015, 更新语法、规则、功能, 浏览器对 ES6 的支持程度并不是很好, 如果写了ES6的代码, 需要运行在浏览器上的时候, 需要将 ES6 的代码转成 ES5 的代码去浏览器上运行。

Babel 是什么

babel 是一个 ES6 转码器, 可以将 ES6 代码转为 ES5 代码, 以便兼容那些还没支持 ES6 的平台

ES6 如何转化为 ES5

- 1、使用npm安装转换插件babel-cli, 在js文件中引入插件
- 2、创建babel-cli配置文件.babelrc, 输入以下内容

```
1 {  
2   "presets": [  
3     "es2015" // 定义转换规则  
4   ],  
5   "plugins": [  
6   ]  
7 }
```

- 3、在终端输入**babel src/index.js -o dist/index.js** (src为开发路径, 即ES6所在目录, dist为转换后ES5路径)

babel-polyfill
Babel默认只转换新的javascript语法, 但并不转换新的API, 比如 Generator、Set、Symbol、promise等全局对象, 以及一些定义在全局对象上的方法都不会转码。如果想让这些方法运行则必须使用babel-polyfill。

安装

```
$ npm install --save babel-polyfill
```

在js中使用

```
require("babel-polyfill");
```

将Babel集成到webpack中
Babel配置
1) 安装babel-loader与babel-core

```
$ npm install babel-core babel-loader --save-dev
```

2)安装预设

```
$ npm install babel-preset-latest --save-dev
```

3)配置.babelrc

```
{"presets":["latest"]}
```

webpack配置

1)安装webpack

```
$ npm install webpack webpack-cli --save-dev
```

2)添加配置文件 webpack.config.js

```
const path=require('path');
module.exports={
  entry: './index.js',
  output:{
    filename:'bundle.js',
    path:path.resolve(__dirname,'dist')
  },
  module:{
    rules:[{
      test:/\.js$/,
      use:'babel-loader'
    }]
  }
}
```

3) 修改package.json

```
"scripts":{
  "build":"webpack"
}
```

4)打包

```
$ npm run build
```

可能会报如下错误:

cdCannot find module '@babel/core' babel-loader@8 requires Babel 7.x. 如果按我上面步骤我们装的babel-loader是8.0.4版本，因为我们只需要重新装7版本。

```
npm install babel-loader@7 --save-dev
```

14、日常前端代码开发中，有哪些值得用 ES6 去改进的编程优化或者规范（必会）

- 1、常用箭头函数来取代 `var self = this;`的做法。
- 2、常用 `let` 取代 `var` 命令。
- 3、常用数组/对象的结构赋值来命名变量，结构更清晰，语义更明确，可读性更好。
- 4、在长字符串多变量组合场合，用模板字符串来取代字符串累加，能取得更好地效果和阅读体验。
- 5、用 `Class` 类取代传统的构造函数，来生成实例化对象。
- 6、在大型应用开发中，要保持 `module` 模块化开发思维，分清模块之间的关系，常用 `import`、`export` 方法。

15、ES6 和 node 的 commonjs 模块化规范的区别（高薪常问）

ES6 是 js 的增强版，是 js 的语法规则，commonjs 都只是为了解决 js 文件之间的依赖和引用问题，所以是一种 js 的包管理规范，其中的代表是 node 遵循 commonjs 规范

16、Promise 中 reject 和 catch 处理上有什么区别（高薪常问）

`reject` 是用来抛出异常，`catch` 是用来处理异常

`reject` 是 `Promise` 的方法，而 `catch` 是 `Promise` 实例的方法

`reject` 后的东西，一定会进入 `then` 中的第二个回调，如果 `then` 中没有写第二个回调，则进入 `catch`

网络异常（比如断网），会直接进入 catch 而不会进入 then 的第二个回调

17、理解 async/await 以及对 Generator 的优势

理解 async await

async await 是用来解决异步的，async 函数是 Generator 函数的语法糖

使用关键字 async 来表示，在函数内部使用 await 来表示异步

async 函数返回一个 Promise 对象，可以使用 then 方法添加回调函数

当函数执行的时候，一旦遇到 await 就会先返回，等到异步操作完成，再接着执行函数体

内 后面的语句

async 较 Generator 的优势

1、内置执行器

Generator 函数的执行必须依靠执行器，而 Async 函数自带执行器，调用方式 跟普通函数的调用一样

2、更好的语义

async 和 await 相较于 * 和 yield 更加语义化

3、更广的适用性

yield 命令后面只能是 Thunk 函数或 Promise 对象，async 函数的 await 后面可以是

Promise 也可以是原始类型的值

4、返回值是 Promise

async 函数返回的是 Promise 对象，比 Generator 函数返回的 Iterator 对象 方便，可以直接使用 then() 方法进行调用

generator 函数就是一个封装的异步任务，也就是异步任务的容器，执行 Generator 函数会返回一个遍历器对象，async 函数的实现，就是将 Generator 函数和自动执行器，包装在一个函数里

18、手写一个 Promise（高薪常问）

```
var Promise = new Promise((resolve, reject) => {  
  if (操作成功) {  
    resolve(value)  
  } else {  
    reject(error)  
  }  
})  
  
Promise.then(function (value) {  
  // success  
}, function (value) {  
  // failure  
})
```

19、Promise 如何封装一个 Ajax（高薪常问）

```
function ajax(optionsOverride) {  
    // 将传入的参数与默认设置合并  
    var options = {};  
    for (var k in ajaxOptions) {  
        options[k] = optionsOverride[k] || ajaxOptions[k];  
    }  
    options.async = options.async === false ? false : true;  
    var xhr = options.xhr = options.xhr || new XMLHttpRequest();  
  
    return new Promise(function (resolve, reject) {  
        xhr.open(options.method, options.url, options.async);  
        xhr.timeout = options.timeout;  
  
        // 设置请求头  
        for (var k in options.headers) {  
            xhr.setRequestHeader(k, options.headers[k]);  
        }  
        // 注册xhr对象事件  
        xhr.onprogress = options.onprogress;  
        xhr.upload.onprogress = options.onuploadprogress;  
        xhr.responseType = options.dataType;  
  
        xhr.onabort = function () {  
            reject(new Error({  
                errorType: 'abort_error',  
                xhr: xhr  
            }));  
        };  
        xhr.ontimeout = function () {  
            reject({  
                errorType: 'timeout_error',  
                xhr: xhr  
            });  
        };  
        xhr.onerror = function () {  
            reject({  
                errorType: 'onerror',  
                xhr: xhr  
            });  
        };  
        xhr.onloadend = function () {  
            if ((xhr.status >= 200 && xhr.status < 300) || xhr.status === 304)  
                resolve(xhr);  
            else  
                reject({  
                    errorType: 'status_error',  
                    xhr: xhr  
                });  
        };  
        try {  
            xhr.send(options.data);  
        } catch (e) {  
            reject({  
                errorType: 'send_error',  
                error: e  
            });  
        }  
    });  
}
```

</script>

20、下面的输出结果是多少（高薪常问）

```
const Promise = new Promise((resolve, reject) => {
```

```
    console.log(2);
```

```
    resolve();
```

```
    console.log(333);
```



```

    })

    Promise.then(() => {

        console.log(666);

    })

    console.log(888);

```

解析：Promise 新建后立即执行，所以会先输出 2, 333，而 Promise.then()内部的代码在 当 次 事件循环的 结尾 立刻执行，所以会继续输出 888，最后输出 666

21、以下代码依次输出的内容是（高薪常问）

```

setTimeout(function () {

    console.log(1)

}, 0);new Promise(function executor(resolve) {

    console.log(2);

    for (var i = 0; i < 10000; i++) {

        i == 9999 && resolve();

    }

    console.log(3);

}).then(function () {

    console.log(4);

});

console.log(5);

```

解析：首先碰到一个 `setTimeout`，于是会先设置一个定时，在定时结束后将传递这个函数 放到任务队列里面，因此开始肯定不会输出 1 。

然后是一个 `Promise`，里面的函数是直接执行的，因此应该直接输出 2 3 。

然后，`Promise` 的 `then` 应当会放到当前 `tick` 的最后，但是还是在当前 `tick` 中。

因此，应当先输出 5，然后再输出 4，最后在到下一个 `tick`，就是 1 。

22、分析下列程序代码，得出运行结果，解释其原因（高薪常问）

```
const Promise = new Promise((resolve, reject) => {  
  console.log(1)  
  resolve()  
  console.log(2)  
})  
Promise.then(() => {  
  console.log(3)  
})  
console.log(4)  
运行结果：1 2 4 3
```

解析：`Promise` 构造函数是同步执行的，`Promise.then` 中的函数是异步执行的。

23、分析下列程序代码，得出运行结果，解释其原因（高薪常问）

```
const Promise = new Promise((resolve, reject) => {
```

```
    resolve('success1')
    reject('error')
    resolve('success2')
  })
  Promise
    .then((res) => {
      console.log('then: ', res)
    })
    .catch((err) => {
      console.log('catch: ', err)
    })
  运行结果: then: success1
```

解析：构造函数中的 resolve 或 reject 只有第一次执行有效，多次调用没有任何作用，

呼 应代码二结论：Promise 状态一旦改变则不能再变。

24、使用结构赋值，实现两个变量的值的交换（高薪常问）

```
let a = 1;let b = 2;
[a,b] = [b,a];
```

25、说一下 ES6 的导入导出模块（高薪常问）

导入模块

通过 import 关键字

// 只导入一个

```
import {sum} from "./example.js"
```

// 导入多个

```
import {sum,multiply,time} from "./exportExample.js"
```

// 导入一整个模块

```
import * as example from "./exportExample.js"
```

导出模块

导出通过 export 关键字

//可以将 export 放在任何变量,函数或类声明的前面

```
export var firstName = 'Chen';
```

```
export var lastName = 'Sunny';
```

```
export var year = 1998;
```

//也可以使用大括号指定所要输出的一组变量

```
var firstName = 'Chen';
```

```
var lastName = 'Sunny';
```

```
var year = 1998;
```

```
export {firstName, lastName, year};
```

//使用 export default 时, 对应的 import 语句不需要使用大括号

```
let bosh = function crs(){}
```

```
export default bosh;
```

```
import crc from 'crc';
```

//不使用 export default 时, 对应的 import 语句需要使用大括号

```
let bosh = function crs(){}
```

```
export bosh;
```

```
import {crc} from 'crc';
```

git

1、git 的基本使用方法（必会）

第一步： window 本机电脑安装 git

第二步： 配置环境变量

安装到 D:\software\git\目录, 把 bin 目录路径完整加入 Path 变量。 D:\software\git\bin

第三步： 配置 git 的 config

```
git config --global user.email "you@example.com"
```

```
git config --global user.name "Your Name"
```

查看你的配置是 `git config --list`

第四步： 使用 git 开始工作

1、在本地建立一个文件夹, 作为本地代码仓库, 并初始化

cmd 中 cd 到该文件夹, 执行 `git init` 命令, 让该文件夹成为受 git 管理的仓库目录。

2、把某个文件添加到本地仓库

执行 `git add HelloWorld.html` 命令

3、提交文件到仓库

`git commit -m "第一次使用 git 提交文件"` 后面的 `" "` 可以写上备注信息的)

2、git 工作流程（必会）

git 的作用

- 1、在工作目录中修改某些文件
- 2、对修改后的文件进行快照，然后保存到暂存区域
- 3、提交更新，将保存在暂存区域的文件快照永久转储到 Git 目录中

git 的工作中使用场景：

两个分支 master 和 dev

项目开始执行流程

`git branch -a` (查看所有分支)

- 0、克隆代码 `git clone 地址`
- 1、拉取线上 master 最新代码: `git pull origin master`
- 2、切换到开发分支: `git checkout dev`
- 3、合并 master 本地分支 (master) : `git merge master`
- 4、开始开发
- 5、开发结束
- 6、查看当前文件更改状态: `git status`
- 7、把所有更改代码放到缓存区: `git add -A`
- 8、查看当前文件更改状态 : `git status`
- 9、缓存区内容添加到仓库中: `git commit -m '本次更改注释'`

10、把代码传到 gitLab 上: `git push origin dev`

11、若代码到达上线标准则合并代码到 master,切换分支到 master: `git checkout master`

12、拉取 master 最新分支: `git pull origin master`

13、合并分支代码到 master(若有冲突则解决冲突): `git merge dev`

14、把当前代码上传到 gitLab: `git push origin master`

15、代码上线后, 用 tag 标签标记发布结点(命名规则: prod_+版本_+上线日期)

`git tag -a prod_V2.1.8_20200701`

16、tag 标签推到 gitLab

`git push origin prod_V2.1.8_20200701`

缓存区的应用

1、需要合并别人代码进来

1.1) 把自己的代码放入暂存: `git stash`

1.2) 如果需要释放出来用: `git stash pop`#恢复最近一次的暂存

1.3) 查看你有哪些队列: `git stash list`

1.4) 删除第一个队列, 以此可以类推: `git stash drop stash@{0}`

2、需要切换分支

2.1) `git add -A`

2.2) `git stash save 'demo'`

2.3) `git stash list`

2.4) `git stash apply stash@{0}`

补充指令

git reflog 查看提交记录命令：

git show # 显示某次提交的内容 git show \$id

git rm <file> # 从版本库中删除文件

git reset <file> # 从暂存区恢复到工作文件

git reset HEAD^ # 恢复最近一次提交过的状态，即放弃上次提交后的所有本次修改

git diff <file> # 比较当前文件和暂存区文件差异 git diff

git log -p <file> # 查看每次详细修改内容的 diff

git branch -r # 查看远程分支

git merge <branch> # 将 branch 分支合并到当前分支

git stash pop git pull # 抓取远程仓库所有分支更新并合并到本地

git push origin master # 将本地主分支推到远程主分支

git branch 分支名#创建分支

git checkout 分支名#切换分支

git checkout -b 分支名#创建并切换分支

git branch --merge / git branch --no-merge#查看已经合并的分支/未合并的分支

git branch -d 分支名 / git branch -D 分支名#删除的已合并的分支/未合并的分支

3、我们如何使用 git 和开源的码云或 github 上面的远端仓库的项目进行工作呢（必会）

客户端本地 git 如何和远程仓库码云，github 连接上次文件

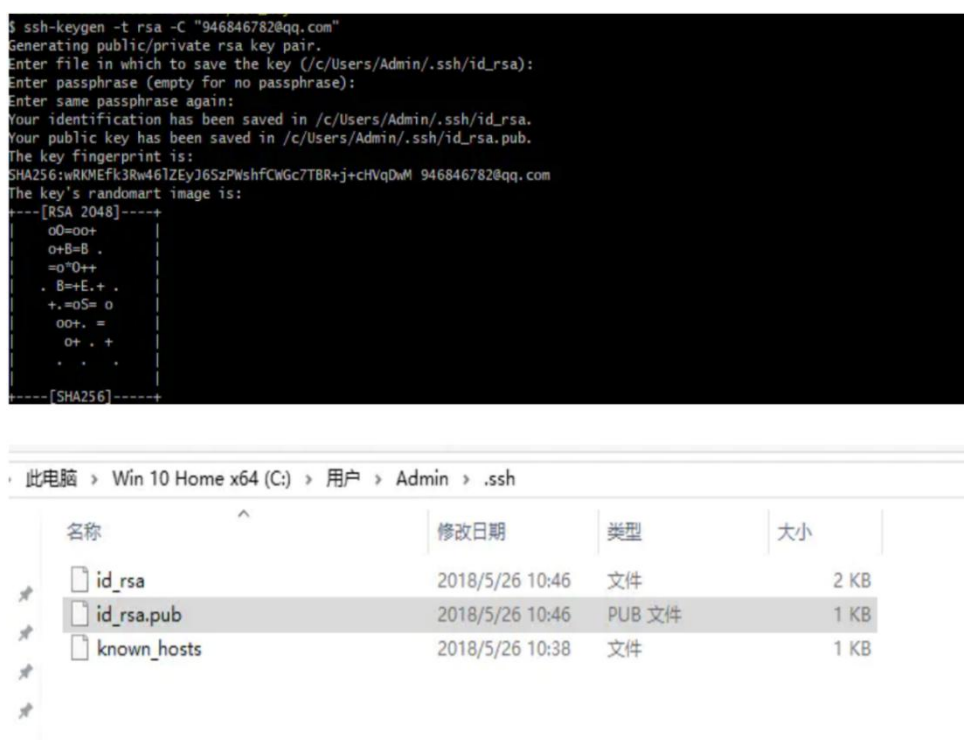
git 仓库如 github 都是通过使用 SSH 与客户端连接的！

我们通过本地 git 生成密钥对后，将公钥保存至 github，每次连接时 SSH 客户端发送本地私钥（默认 ~/.ssh/id_rsa）到服务端验证。单用户情况下，连接的服务器上保存的公钥和发送的私钥自然是配对的

命令如下：ssh-keygen -t rsa -C 'XXX@qq.com' -f id_rsa_second

或 ssh-keygen -t rsa -C "XXX@qq.com"

邮件可以换成你的



添加公钥（id_rsa_second.pub）到你的远程仓库（github）

登陆你的 github 帐户。点击你的头像，然后 Settings -> 左栏点击 SSH and GPG keys

-> 点击 New SSH key

然后你复制上面的公钥内容，粘贴进“Key”文本域内。title 域，自己随便起个名字。

点击 Add key。

完成以后，验证下这个 key 是不是正常工作：

```
$ ssh -T git@github.com
```

Attempts to ssh to github

如果，看到：

```
Hi xxx! You've successfully authenticated, but github does not # provide shell
```

access.

表示设置已经成功

码云：

进入码云的设置页面

码云 开源软件 企业版 博客 我的码云 搜索项目、代码片段...

xiaomai 加入时间 3年前

基本信息 个人地址 第三方登录

注册邮箱 946846782@qq.com 修改邮箱

手机号码 绑定号码

昵称 xiaomai

微博 微博

博客 博客

自我介绍

更新

基本设置

- 个人资料
- 修改密码
- 个性地址
- 通知设置

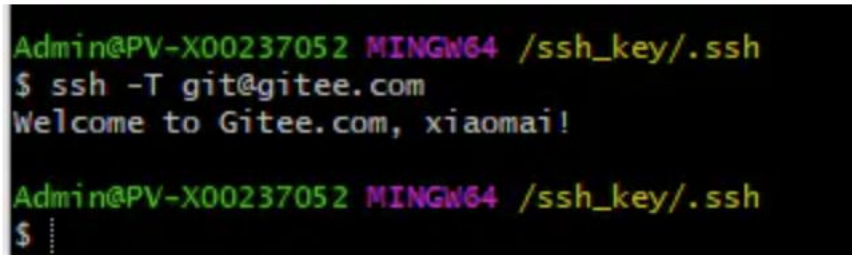
安全设置

- 私人令牌
- SSH公钥

数据管理

- 升级为组织
- 升级为企业版
- 私有项目成员
- 代码风格
- 第三方应用
- 捐赠管理

在终端 (Terminal) 中输入 `ssh -T git@gitee.com` 若返回 `Welcome to gitee.com,`
`yourname!`

A terminal window with a black background and green text. The prompt is 'Admin@PV-X00237052 MINGW64 /ssh_key/.ssh'. The user enters '\$ ssh -T git@gitee.com'. The output is 'Welcome to Gitee.com, xiaomai!'. The user enters another '\$' and the prompt returns.

代表成功!

通常步骤:

本地新建仓库, 输入 `git init` 初始化, 让 `git` 接管

关联一个远程仓库: `git remote add origin git@github.com:XXXXXXXXXX.git`

把文件添加到本地版本库

`git add` 文件名

把文件修改提交到本地仓库

`git commit -m"注释"`

`git pull origin master` 先将 `github` 上的代码 `pull` 下来

然后在 `git push origin master` 将最新的修改推送到远程仓库

`git -` 查看远程仓库信息

可以通过命令 `git remote show [remote-name]` 查看某个远程仓库的详细信息

4、git、github、gitlab 三者之间的联系以及区别 (必会)

1、git

git 是一个版本控制系统。

版本控制是一种用于记录一个或多个文件内容变化，方便我们查阅特定版本修订情况的系统。

早期出现的版本控制系统有：svn、cvs 等，它们是集中式版本控制系统，都有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同合作的开发人员都通过客户端连接到这台服务器，取出最新的文件或者提交更新。

而我们的主角 git 是分布式版本控制系统。git 已经成为越来越多开发者的青睐，因为分布式的优势是很显著的。

2、集中式和分布式版本控制系统的区别：

2.1) 分布式版本控制系统下的本地仓库包含代码库还有历史库，在本地就可以查看版本历史

2.2) 而集中式版本控制系统下的历史仓库是存在于中央仓库，每次对比与提交代码都必须连接到中央仓库

2.3) 多人开发时，如果充当中央仓库的 git 仓库挂掉了，任何一个开发者都可以随时创建一个新的中央仓库然后同步就可以恢复中央仓库

3、github 和 gitlab

github 和 gitlab 都是基于 web 的 git 仓库，使用起来二者差不多，它们都提供了分享开源项目的平台，为开发团队提供了存储、分享、发布和合作开发项目的中心化云存储的场所。

github 作为开源代码库，拥有超过 900 万的开发者用户，目前仍然是最火的开源项目托管平台，github 同时提供公共仓库和私有仓库，但如果使用私有仓库，是需要付费的。

gitlab 解决了这个问题，你可以在上面创建私人的免费仓库。

gitlab 让开发团队对他们的代码仓库拥有更多的控制，相比较 github，

gitlab 特色

- 3.1) 允许免费设置仓库权限；
- 3.2) 允许用户选择分享一个 project 的部分代码；
- 3.3) 允许用户设置 project 的获取权限，进一步提升安全性；
- 3.4) 可以设置获取到团队整体的改进进度；
- 3.5) 通过 innersourcing 让不在权限范围内的人访问不到该资源；

所以，从代码的私有性上来看，gitlab 是一个更好的选择。但是对于开源项目而言，github 依然是代码托管的首选。

5、github 和码云的区别（必会）

github

全英文、用户基数多，知名库多、国内访问的话，偶尔会有不稳定，出现上不去的情况、

私有项目需要付费

码云

全中文、用户量没有 github 多，知名库相对较少、服务器在国内，相对稳定、每个用户有

1000 个免费的私有项目、访问速度很快，支持 svn，git 两种方式、每个仓库有 1G 的容量

限 制

6、提交时发生冲突，你能解释冲突是如何产生的吗？你是如何解决的（必会）

冲突是如何产生

开发过程中，我们都有自己的特性分支，所以冲突发生的并不多，但也碰到过。诸如公共类的公共方法，我和别人同时修改同一个文件，他提交后我再提交就会报冲突的错误。

如何解决冲突

1、发生冲突，在 IDE 里面一般都是对比本地文件和远程分支的文件，然后把远程分支上文件的内容手工修改到本地文件，然后再提交冲突的文件使其保证与远程分支的文件一致，这样才会消除冲突，然后再提交自己修改的部分。特别要注意下，修改本地冲突文件使其与远程仓库的文件保持一致后，需要提交后才能消除冲突，否则无法继续提交。必要时可与同事交流，消除冲突。

2、发生冲突，也可以使用命令

通过 git stash 命令，把工作区的修改提交到栈区，目的是保存工作区的修改；

通过 git pull 命令，拉取远程分支上的代码并合并到本地分支，目的是消除冲突；

通过 git stash pop 命令，把保存在栈区的修改部分合并到最新的工作空间中；

分支提交冲突：当分支对某文件某句话进行修改后，切换到主分支也对该文件该句话进行修改，使用 git merge 进行合并，需要将两个修改进行合并。此时合并产生冲突

3、另外一种解决方法

3.1) git status 查看冲突文件

3.2) 编辑器打开冲突文件，查看内容。Git 用<<<<<<, =====, >>>>>>

标记出不同分支的内容

3.3) 修改文件内容

3.4) 提交 `git add file ; git commit -m ""`

查看分支合并图 `git log --graph`

7、如果本次提交误操作，如何撤销（必会）

如果想撤销提交到索引区的文件，可以通过 `git reset HEAD file`

如果想撤销提交到本地仓库的文件

可以通过 `git reset --soft HEAD^n` 恢复当前分支的版本库至上一次提交的状态，索引区和工作空间不变更；可以通过 `git reset --mixed HEAD^n` 恢复当前分支的版本库和索引区至上一次提交的状态，工作区不变更；可以通过 `git reset --hard HEAD^n` 恢复当前分支的版本库、索引区和工作空间至上一次提交的状态。

8、git 修改提交的历史信息（必会）

git 修改提交的历史信息详细操作

`git rebase -i HEAD~3`

输出如下

pick 1 commit 1

pick 2 commit 2

pick 3 commit 3

要修改哪个，就把那行的 pick 改为 edit，然后退出。例如想修改 commit 1 的 author，光标移到第一个 pick，按 i 键进入 INSERT 模式，把 pick 改为 edit：

```
edit 1 commit 1
```

```
pick 2 commit 2
```

```
pick 3 commit 3
```

```
...
```

```
- INSERT -
```

然后按 esc 键，退出 INSERT 模式，输入:wq 退出，这时可以看到提示，可以修改 commit 1 的信息了

输入 amend 命令重置用户信息： `$ git commit --amend --reset-author`

会出现 commit 1 的提交记录及注释内容，可进入 INSERT 模式修改注释，:wq 退出

这时再查看提交历史，发现 commit 1 的 author 已经变成 b(b@email.com)了，且是最新一次记录

通过 continue 命令回到正常状态： `$ git rebase --continue`

9、如何删除 github 和 gitlab 上的文件夹（必会）

解决办法

重点在于 `git push -u`

方法一：

这里以删除 .setting 文件夹为案例

`git rm -r --cached .setting` #--cached 不会把本地的.setting 删除

```
git commit -m 'delete .setting dir'
```

```
git push -u origin master
```

方法二:

如果误提交的文件夹比较多, 方法一也较繁琐

直接修改.gitignore 文件,将不需要的文件过滤掉, 然后执行命令

```
git rm -r --cached .
```

```
git add .
```

```
git commit
```

```
git push -u origin master
```

10、如何查看分支提交的历史记录? 查看某个文件的历史记录呢 (必会)

查看分支的提交历史记录

命令 `git log -number`: 表示查看当前分支前 `number` 个详细的提交历史记录

命令 `git log -number --pretty=oneline`: 在上个命令的基础上进行简化, 只显示 sha-1

码和提交信息;

命令 `git reflog -number`: 表示查看所有分支前 `number` 个简化的提交历史记录

命令 `git reflog -number --pretty=oneline`: 显示简化的信息历史信息

如果要查看某文件的提交历史记录, 直接在上面命令后面加上文件名即可

注意：如果没有 number 则显示全部提交次数

11、git 跟 svn 有什么区别（必会）

git 是分布式版本控制系统，其他类似于 svn 是集中式版本控制系统。

分布式区别于集中式在于：每个节点的地位都是平等，拥有自己的版本库，在没有网络的情况下，对工作空间内代码的修改可以提交到本地仓库，此时的本地仓库相当于集中式的远程仓库，可以基于本地仓库进行提交、撤销等常规操作，从而方便日常开发

git 和 svn 的区别

git 是分布式版本控制，svn 是集中式版本控制（核心区别）

git 相对于 svn 的优势就是不需要网络即可版本控制

git 把内容按数据方式存储，而 svn 是按文件

git 可以是公用的，可以分享，svn 基本是公司内部才能访问，网外不方便访问

git 不依赖中央服务器，即使服务器有问题也不受影响，svn 依赖服务器，一旦服务器有问题 就会受影响

git 没有一个全局的版本号，svn 有

12、我们在本地工程常会修改一些配置文件，这些文件不需要被提交，而我们又不想每次执行 git status 时都让这些文件显示出来，我们该如何操作（必会）

首先利用命令 touch .gitignore 新建文件

```
$ touch .gitignore
```

然后往文件中添加需要忽略哪些文件夹下的什么类型的文件

```
$ vim .gitignore
```

```
$ cat .gitignore
```

```
/target/class
```

```
.settings
```

```
.imp
```

```
*.ini
```

注意：忽略/target/class 文件夹下所有后缀名为.settings, .imp 的文件，忽略所有后缀名为.ini 的文件。

13、git fetch 和 git merge 和 git pull 的区别（必会）

区别如下

git pull 相当于 git fetch 和 git merge，即更新远程仓库的代码到本地仓库，然后将内容合并到当前分支。

git merge: 将内容合并到当前分支

git pull 相当于是从远程获取最新版本并 merge 到本地

命令从中央存储库中提取特定分支的新更改或提交，并更新本地存储库中的目标分支。

git fetch 相当于是从远程获取最新版本到本地，不会自动 merge

也用于相同的目的，但它的工作方式略有不同。当你执行 git fetch 时，

它会从所需的分支中提取所有新提交，并将其存储在本地存储库中的新分支中。如果要在

目标分支中反映这些更改，必须在 git fetch 之后执行 git merge。只有在对目标分支和获

取的分支进行合并后才会更新目标分支。

为了方便起见，请记住以下等式：

`git pull = git fetch + git merge`