第三版：前端 40 道

var、let、const 之间的区别

1、 var 声明变量可以重复声明，而 let 不可以重复声明

2、 var 是不受限于块级的，而 let 是受限于块级

3、 var 会与 window 相映射（会挂一个属性），而 let 不与 window 相映射

4、 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

5、 const 声明之后必须赋值，否则会报错

6、 const 定义不可变的量，改变了就会报错

7、 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

解构赋值

数组解构

```
let [a, b, c] = [1, 2, 3]     //a=1, b=2, c=3
let [d, [e], f] = [1, [2], 3]      //嵌套数组解构 d=1, e=2, f=3
let [g, ...h] = [1, 2, 3]     //数组拆分 g=1, h=[2, 3]
let [i,,j] = [1, 2, 3]     //不连续解构 i=1, j=3
```


1

```
let [k,l] = [1, 2, 3]    //不完全解构 k=1, l=2
```

对象解构

```
let {a, b} = {a: 'aaaa', b: 'bbbb'}          //a='aaaa' b='bbbb'
let obj = {d: 'aaaa', e: {f: 'bbbb'}}
let {d, e:{f}} = obj       //嵌套解构 d='aaaa' f='bbbb'
let g;
(g = {g: 'aaaa'})    //以声明变量解构 g='aaaa'
let [h, i, j, k] = 'nice'      //字符串解构 h='n' i='i' j='c' k='e'
```

函数参数的定义

一般我们在定义函数的时候，如果函数有多个参数时，在 es5 语法中函数调用时参数必须一一对应，否则就会出现赋值错误的情况，来看一个例子：

```
function personInfo(name, age, address, gender) {
  console.log(name, age, address, gender)
}
personInfo('william', 18, 'changsha', 'man')
```

上面这个例子在对用户信息的时候需要传递四个参数，且需要一一对应，这样就会极易出现参数顺序传错的情况，从而导致 bug，接下来来看 es6 解构赋值是怎么解决这个问题的：

```
function personInfo({name, age, address, gender}) {
  console.log(name, age, address, gender)
}
personInfo({gender: 'man', address: 'changsha', name: 'william', age: 18})
```

这么写我们只知道要传声明参数就行来，不需要知道参数的顺序也没关系

交换变量的值

在 es5 中我们需要交换两个变量的值需要借助临时变量的帮助，来看一个例子：

```
var a=1, b=2, c
c = a
a = b
b = c
console.log(a, b)
```

来看 es6 怎么实现：

```
let a=1, b=2;
[b, a] = [a, b]
console.log(a, b)
```

是不是比 es5 的写法更加方便呢

函数默认参数

在日常开发中，经常会有这种情况：函数的参数需要默认值，如果没有默认值在使用的时候就会报错，来看 es5 中是怎么做的：

```
function saveInfo(name, age, address, gender) {
  name = name || 'william'
  age = age || 18
  address = address || 'changsha'
  gender = gender || 'man'
  console.log(name, age, address, gender)
}
```

```
saveInfo()
```

在函数离 main 先对参数做一个默认值赋值，然后再使用避免使用的过程中报错，再来看 es6 中的使用的方法：

```
function saveInfo({name= 'william', age= 18, address= 'changsha',
gender= 'man'} = {}) {
  console.log(name, age, address, gender)
}
saveInfo()
```

在函数定义的时候就定义了默认参数，这样就免了后面给参数赋值默认值的过程，是不是看起来简单多了

forEach、for in、for of 三者区别

1、 forEach 更多的用来遍历数

2、 for in 一般常用来遍历对象或 json

3、 for of 数组对象都可以遍历，遍历对象需要通过和 Object.keys()

4、 for in 循环出的是 key，for of 循环出的是 value

使用箭头函数应注意什么?

1、 用了箭头函数，this 就不是指向 window，而是父级（指向是可变的）

2、 不能够使用 arguments 对象

3、 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

4、 不可以使用 yield 命令，因此箭头函数不能用作 Generator 函数

Set、Map 的区别

应用场景 Set 用于数据重组，Map 用于数据储存

Set:

1、 成员不能重复
2、 只有键值没有键名，类似数组
3、 可以遍历，方法有 add, delete,has

Map:

1、 本质上是健值对的集合，类似集合
2、 可以遍历，可以跟各种数据格式转换

promise 对象的用法,手写一个 promise

promise 是一个构造函数，下面是一个简单实例

```
var promise = new Promise((resolve,reject) => {
    if (操作成功) {
        resolve(value)
    } else {
        reject(error)
```

```
    }
})
promise.then(function (value) {
    // success
},function (value) {
    // failure
})
```

如何创建一个 ajax

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

同步和异步的区别

同步：

浏览器访问服务器请求，用户看得到页面刷新，重新发请求,等请求完，页面刷新，新内容出现，用户看到新内容,进行下一步操作

异步：

浏览器访问服务器请求，用户正常操作，浏览器后端进行请求。等请求完，页面不刷新，新内容也会出现，用户看到新内容

ajax 的优点和缺点

ajax 的优点

1、 无刷新更新数据（在不刷新整个页面的情况下维持与服务器通信）

2、 异步与服务器通信（使用异步的方式与服务器通信，不打断用户的操作）

3、 前端和后端负载均衡（将一些后端的工作交给前端，减少服务器与宽度的负担）

4、 界面和应用相分离（ajax 将界面和应用分离也就是数据与呈现相分离）

ajax 的缺点

1、 ajax 不支持浏览器 back 按钮

2、 安全问题 Aajax 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 Back 与 History 后退按钮的正常行为等浏览器机制

get 和 post 的区别

1、 get 和 post 在 HTTP 中都代表着请求数据，其中 get 请求相对来说更简单、快速，效率高些

2、 get 相对 post 安全性低

3、 get 有缓存，post 没有

4、 get 体积小，post 可以无限大

5、 get 的 url 参数可见，post 不可见

6、 get 只接受 ASCII 字符的参数数据类型，post 没有限制

7、 get 请求参数会保留历史记录，post 中参数不会保留

8、 get 会被浏览器主动 catch，post 不会，需要手动设置

9、 get 在浏览器回退时无害，post 会再次提交请求

什么时候使用 post?

post 一般用于修改服务器上的资源，对所发送的信息没有限制。比如

1、 无法使用缓存文件（更新服务器上的文件或数据库）

2、 向服务器发送大量数据（POST 没有数据量限制）

3、 发送包含未知字符的用户输入时，POST 比 GET 更稳定也更可靠

如何解决跨域问题

跨域的概念：协议、域名、端口都相同才同域，否则都是跨域

解决跨域问题：

1、 使用 JSONP（json+padding）把数据内填充起来

2、 CORS 方式（跨域资源共享），在后端上配置可跨域

3、 服务器代理，通过服务器的文件能访问第三方资源

什么是 Ajax 和 JSON，它们的优点和缺点

Ajax：

Ajax 是异步 JavaScript 和 XML，用于在 Web 页面中实现异步数据交互

Ajax 优点：

异步请求响应快，用户体验好；页面无刷新、数据局部更新；按需取数据，减少了冗余请求和服务器的负担；

Ajax 缺点：

异步回调问题、this 指向问题、路由跳转 back 问题；对搜索引擎的支持比较弱，对于一些手机还不是很好的支持

JSON：

是一种轻量级的数据交换格式，看着像对象，本质是字符串

JSON 优点：

轻量级、易于人的阅读和编写，便于 js 解析，支持复合数据类型

JSON 缺点：

没有 XML 格式这么推广的深入人心和使用广泛，没有 XML 那么通用性

git 常用的命令

1、 从远程库克隆到本地：git clone 网站上的仓库地址

2、 新增文件的命令：git add .

3、 提交文件的命令：git commit –m 或者 git commit –a

4、 查看工作区状况：git status –s

5、 拉取合并远程分支的操作：git fetch/git merge 或者 git pull

6、 查看提交记录命令：git reflog

webpack 打包原理

webpack 只是一个打包模块的机制，只是把依赖的模块转化成可以代表这些包的静态文件。webpack 就是识别你的 入口文件。识别你的模块依赖，来打包你的代码。至于你的代码使用的是 commonjs 还是 amd 或者 es6 的 import。webpack 都会对其进行分析。来获取代码的依赖。webpack 做的就是分析代码。转换代码，

编译代码，输出代码。webpack 本身是一个 node 的模块，所以 webpack.config.js 是以 commonjs 形式书写的(node 中的模块化是 commonjs 规范的)

模块热更新

模块热更新是 webpack 的一个功能，他可以使代码修改过后不用刷新就可以更新，是高级版的自动刷新浏览器

devServer 中通过 hot 属性可以控制模块的热替换

通过配置文件

```
const webpack = require('webpack');
const path = require('path');
let env = process.env.NODE_ENV == "development" ? "development" : "production";
const config = {
  mode: env,
 devServer: {
      hot:true
 }
}
  plugins: [
      new webpack.HotModuleReplacementPlugin(), //热加载插件
  ],
module.exports = config;
```

通过命令行

```
"scripts": {
```


11

```
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "NODE_ENV=development   webpack-dev-server --config
webpack.develop.config.js --hot",
  },
```

如何提高 webpack 构建速度

1、 通过 externals 配置来提取常用库

2、 利用 DllPlugin 和 DllReferencePlugin 预编译资源模块 通过 DllPlugin 来对那些我们引用但是绝对不会修改的 npm 包来进行预编译，再通过 DllReferencePlugin 将预编译的模块加载进来

3、 使用 Happypack 实现多线程加速编译

要注意的第一点是，它对 file-loader 和 url-loader 支持不好，所以这两个 loader 就不需要换成 happypack 了，其他 loader 可以类似地换一下

4、 使用 Tree-shaking 和 Scope Hoisting 来剔除多余代码 5、使用 fast-sass-loader 代替 sass-loader 6、babel-loader 开启缓存

babel-loader 在执行的时候，可能会产生一些运行期间重复的公共文件，造成代码体积大冗余，同时也会减慢编译效率 可以加上 cacheDirectory 参数或使用 transform-runtime 插件试试

```
// webpack.config.js
use: [{
    loader: 'babel-loader',
    options: {
        cacheDirectory: true
```


12

```
}]
// .bablerc
{
    "presets": [
        "env",
        "react"
    ],
    "plugins": ["transform-runtime"]
}
```

不需要打包编译的插件库换成全局"script"标签引入的方式

比如 jQuery 插件，react, react-dom 等，代码量是很多的，打包起来可能会很耗时 可以直接用标签引入，然后在 webpack 配置里使用 expose-loader 或 externals 或 ProvidePlugin 提供给模块内部使用相应的变量

```
// @1
use: [{
    loader: 'expose-loader',
    options: '$'
    }, {
    loader: 'expose-loader',
    options: 'jQuery'
    }]
// @2
externals: {
        jquery: 'jQuery'
    },
// @3
        new webpack.ProvidePlugin({
            $: 'jquery',
```


13

```
            jQuery: 'jquery',
            'window.jQuery': 'jquery'
        }),
```

优化构建时的搜索路径

在 webpack 打包时，会有各种各样的路径要去查询搜索，我们可以加上一些配置，让它搜索地更快 比如说，方便改成绝对路径的模块路径就改一下，以纯模块名来引入的可以加上一些目录路径 还可以善于用下 resolve alias 别名 这个字段来配置 还有 exclude 等的配置，避免多余查找的文件，比如使用 babel 别忘了剔除不需要遍历的

webpack 的优点

专注于处理模块化的项目，能做到开箱即用，一步到位

可通过 plugin 扩展，完整好用又不失灵活

使用场景不局限于 web 开发

社区庞大活跃，经常引入紧跟时代发展的新特性，能为大多数场景找到已有的开源扩展

良好的开发体验

webpack 的缺点

webpack 的缺点是只能用于采用模块化开发的项目

微信小程序，文件主要目录及文件作用

```
- component ———————————————————— 组件文件夹
  - navBar                  —— 底部组件
    - navBar.js             —— 底部组件的 JS 代码
    - navBar.json           —— 底部组件的配置文件
    - navBar.wxml           —— 底部组件的 HTML 代码
    - navBar.wxss           —— 底部组件的 CSS 代码
- pages ———————————————————————— 页面文件夹
  - index                   —— 首页
    - index.js              —— 首页的 JS 代码
    - index.json            —— 首页的配置文件
    - index.wxml            —— 首页的 HTML 代码
    - index.wxss            —— 首页的 CSS 代码
- public ———————————————————————— 图片文件夹
- utils ————————————————————————— 工具文件夹
  - api.js                  —— 控制 API 的文件
  - md5.js                  —— 工具 - MD5 加密文件
  - timestamp.js            —— 工具 - 时间戳文件
- app.json ———————————————————— 设置全局的基础数据等
- app.wxss ———————————————————— 公共样式，可通过 import 导入更多
- project.config.json ———————— 项目配置文件
```

微信小程序，生命周期

onLoad()：页面加载时触发。

onShow()：页面显示/切入前台时触发。

15

onReady()：页面初次渲染完成时触发。

onHide()：页面隐藏/切入后台时触发。

onUnload()：页面卸载时触发。

微信小程序，如何封装数据请求

封装接口

项目/utils/api.js

```
// 将请求进行 Promise 封装
const fetch = ({url, data}) => {

  // 打印接口请求的信息
  console.log(`【step 1】API 接口：${url}`);
  console.log("【step 2】data 传参：");
  console.log(data);

  // 返回 Promise
  return new Promise((resolve, reject) => {
    wx.request({
      url: getApp().globalData.api + url,
      data: data,
      success: res => {

        // 成功时的处理
        if (res.data.code == 0) {
          console.log("【step 3】请求成功：");
          console.log(res.data);
          return resolve(res.data);
```

```
          } else {
            wx.showModal({
              title: '请求失败',
              content: res.data.message,
              showCancel: false
            });
          }

        },
        fail: err => {
          // 失败时的处理
          console.log(err);
          return reject(err);
        }
      })
  })

}

/**
 * code 换取 openId
 * @data {
 *    jsCode - wx.login() 返回的 code
 * }
 */
export const wxLogin = data => {
  return fetch({
    url: "tbcUser/getWechatOpenId",
    data: data
  })
}
```


17

调用接口

项目/pages/login/login.js

```
import {
  wxLogin,
} from '../../utils/api.js'
```

使用接口

项目/pages/login/login.js

```
wxLogin({
  jsCode: this.data.code
}).then(
  res => {
    console.log("【step 4】返回成功处理：");
    console.log(res.data);
  },
  err => {
    console.log("【step 4】返回失败处理：");
    console.log(err);
  }
)
```

微信小程序，页面数据传递

通过 url 携带参数，在 onLoad() 中通过 options 获取 url 上的参数：

18

```
<navigator url="../index/index?userId={{userId}}"></navigator>

<!-- 这两段是分别在 HTML 和 JS 中的代码 -->

onLoad: function(options) {
  console.log(options.userId);
}
```

通过 Storage 来传递参数：

```
wx.setStorageSync('userId', 'jsliang');
wx.getStorageSync('userId');
```

WXML 传递数据到 JS

login.wxml

```
<text bindtap="clickText" data-labelId="{{userId}}">点击传递数据到JS</text>
```

login.js

```
clickText(e) {
  console.log(e.currentTarget.labelid)
}
```

组件调用传参

组件接收数据：component-tag-name

```
Component({
```

```
  properties: {
    // 这里定义了 innerText 属性，属性值可以在组件使用时指定
    innerText: {
      type: String,
      value: 'default value',
    }
  }
})
```

使用组件的页面定义 json

```
{
  "usingComponents": {
    "component-tag-name": "../component/component"
  }
}
```

使用组件的页面 HTML 代码

```
<view>
  <!-- 以下是对一个自定义组件的引用 -->
  <component-tag-name inner-text="Some
text"></component-tag-name>
</view>
```

通过接口调用传递参数

微信小程序，加载性能优化方法

1、 通过 this.$preload() 预加载用户可能点击的第二个页面

2、 组件化页面，出现两次以上的部分都进行封装成组件

3、 提取共用的 CSS 样式

4、 优化图片：TinyPNG

微信小程序与原生 APP、Vue、H5 差异

微信小程序优势

1、 无需下载

2、 打开速度较快

3、 开发成本低于原生 APP

微信小程序劣势

1、 限制多。页面大小不能超过 1M，不能打开超过 5 个层级的页面

2、 样式单一。小程序内部组件已经成宿，样式不可以修改

3、 推广面窄。跑不出微信，还不能跑入朋友圈

微信小程序 VS 原生 APP

微信小程序有着低开发成本、低获客成本、无需下载的优势

微信小程序 VS H5

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题
21

1、 依赖环境不同。一个能在多种手机浏览器运行。一个只能在微信中的非完整的浏览器

2、 开发成本不同。一个可能在各种浏览器出问题。一个只能在微信中运行

微信小程序 VS Vue

微信小程序看似就是阉割版的 Vue

微信小程序原理

本质上就是一个单页面应用，所有的页面渲染和事件处理，都在一个页面中进行

架构为数据驱动的模式，UI 和数据分离，所有页面的更新，都需要通过对数据的更改来实现

微信小程序分为两个部分：webview 和 appService。其中 webview 主要用来展示 UI，appServer 用来处理业务逻辑、数据及接口调用。它们在两个进程中进行，通过系统层 JSBridge 实现通信，实现 UI 的渲染、事件的处理

wxml 与标准的 html 的异同

wxml 基于 xml 设计，标签只能在微信小程序中使用，不能使用 html 的标签

网络分层

目前网络分层可分为两种：OSI 模型和 TCP/IP 模型

22

OSI 模型

1、 应用层（Application）

2、 表示层（Presentation）

3、 会话层（Session）

4、 传输层（Transport）

5、 网络层（Network）

6、 数据链路层（Data Link）

7、 物理层（Physical）

TCP/IP 模型

1、 应用层（Application）

2、 传输层（Host-to-Host Transport）

3、 互联网层（Internet）

4、 网络接口层（Network Interface）

HTTP/HTTPS

1、 https 协议需要到 ca 申请证书，一般免费证书较少，因而需要一定费用

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。

HTTP 状态码

区分状态码

1××开头 - 信息提示
2××开头 - 请求成功
3××开头 - 请求被重定向
4××开头 - 请求错误
5××开头 - 服务器错误

常见状态码

200 - 请求成功，Ajax 接受到信息了
400 - 服务器不理解请求
403 - 服务器拒绝请求
404 - 请求页面错误
500 - 服务器内部错误，无法完成请求

HTML 优化

1、 避免 HTML 中书写 CSS 代码，因为这样难以维护。

2、 使用 Viewport 加速页面的渲染。

3、 使用语义化标签，减少 CSS 代码，增加可读性和 SEO。

4、 减少标签的使用，DOM 解析是一个大量遍历的过程，减少不必要的标签，能降低遍历的次数。

5、 避免 src、href 等的值为空，因为即时它们为空，浏览器也会发起 HTTP 请求。

6、 减少 DNS 查询的次数

CSS 优化

1、 优化选择器路径：使用 .c {} 而不是 .a .b .c {}。

2、 选择器合并：共同的属性内容提起出来，压缩空间和资源开销。

3、 精准样式：使用 padding-left: 10px 而不是 padding: 0 0 0 10px。

4、 雪碧图：将小的图标合并到一张图中，这样所有的图片只需要请求一次。

5、 避免通配符：.a .b *{} 这样的选择器，根据从右到左的解析顺序在解析过程中遇到通配符* {} 6、会遍历整个 DOM，性能大大损耗。

7、 少用 float：float 在渲染时计算量比较大，可以使用 flex 布局。

8、 为 0 值去单位：增加兼容性。

9、 压缩文件大小，减少资源下载负担。

JavaScript 优化

1、 尽可能把<script> 标签放在 body 之后，避免 JS 的执行卡住 DOM 的渲染，最大程度保证页面尽快地展示出来

2、 尽可能合并 JS 代码：提取公共方法，进行面向对象设计等……

3、 CSS 能做的事情，尽量不用 JS 来做，毕竟 JS 的解析执行比较粗暴，而 CSS 效率更高。

4、 尽可能逐条操作 DOM，并预定好 CSs 样式，从而减少 reflow 或者 repaint 的次数。

5、 尽可能少地创建 DOM，而是在 HTML 和 CSS 中使用 display: none 来隐藏，按需显示。

6、 压缩文件大小，减少资源下载负担。

面试常问

1、 自我介绍

2、 你的项目中技术难点是什么？遇到了什么问题？你是怎么解决的？

3、 你认为哪个项目做得最好？

4、 平时是如何学习前端开发的？

5、 你最有成就感的一件事

6、 你是怎么学习前端的

面试人事面

1、 面试完你还有什么问题要问的吗

2、 你有什么爱好？

3、 你最大的优点和缺点是什么？

4、 你为什么会选择这个行业，职位？

5、 你觉得你适合从事这个岗位吗？

6、 你有什么职业规划？

7、 你对工资有什么要求？

8、 如何看待前端开发？

9、 未来三到五年的规划是怎样的？

谈谈你对重构的理解

关注公众号：磊哥聊编程，回复：面试题，获取最新版面试题 27

网络重构：在不改变外部行为的前提下，简化结构、添加可读性，而在网站前端保持一致的行为。也就是说是在不改变 UI 的情况下，对网站进行优化， 在扩展的同时保持一致的 UI

对于传统的网站来说重构通常是：

1、 表格(table)布局改为 DIV+CSS

2、 使网站前端兼容于现代浏览器(针对于不合规范的 CSS、如对 IE6 有效的)

3、 对于移动平台的优化

4、 针对于 SEO 进行优化

什么样的前端代码是好的？

高复用低耦合，这样文件小，好维护，而且好扩展

对前端工程师这个职位是怎么样理解的？它的前景会怎么样？

前端是最贴近用户的程序员，比后端、数据库、产品经理、运营、安全都近

1、 实现界面交互
2、 提升用户体验
3、 有了 Node.js，前端可以实现服务端的一些事情

前端是最贴近用户的程序员，前端的能力就是能让产品从 90 分进化到 100 分，甚至更好，

与团队成员，UI 设计，产品经理的沟通；

做好的页面结构，页面重构和用户体验；

你觉得前端工程的价值体现在哪?

1、 为简化用户使用提供技术支持（交互部分）

2、 为多个浏览器兼容性提供支持

3、 为提高用户浏览速度（浏览器性能）提供支持

4、 为跨平台或者其他基于 webkit 或其他渲染引擎的应用提供支持

5、 为展示数据提供支持（数据接口）

平时如何管理你的项目?

1、 先期团队必须确定好全局样式（globe.css），编码模式(utf-8) 等；

2、 编写习惯必须一致（例如都是采用继承式的写法，单样式都写成一行）；

3、 标注样式编写人，各模块都及时标注（标注关键样式调用的地方）；

4、 页面进行标注（例如 页面 模块 开始和结束）；

5、 CSS 跟 HTML 分文件夹并行存放，命名都得统一（例如 style.css）；

6、 JS 分文件夹存放 命名以该 JS 功能为准的英文翻译。

7、 图片采用整合的 images.png png8 格式文件使用 - 尽量整合在一起使用方便将来的管理

29

移动端（Android IOS）怎么做好用户体验？

1、 清晰的视觉纵线、

2、 信息的分组、极致的减法、

3、 利用选择代替输入、

4、 标签及文字的排布方式、

5、 依靠明文确认密码、

6、 合理的键盘利用

Process finished with exit code 0