一、回顾 & 补充

1. Promise

1) promise 应用：所有异步场景均可用

2) 执行流程　　微任务与宏任务为准

E F G A H C B I D

{ 宏任务什么都没做，仍有一些
微任务，将任务移到本轮最后。

消息队列 / 回调队列 / 任务队列

3) promise.all()

同构：既可以在node中执行也可在浏览器中执行

npx：npm 5.2之后的小工具：用于执行远程模块 → 如果本地无此模块，会自动下载
用于执行 node_module/bin下的指令

希望单个错误不影响整体，则在每个 promise 中都 catch 一下。

promises = urls.map(item => axios(item).catch(e=>{}))

promise.all(promises)　　catch一下
每个都是成功的

⊛ promise.all 中其中一个失败，并不会影响其他 promise 的执行（不会中断）

4) promise.allSettled(...) : ES2020

不会因为一个错，就失败，永远成功。

{status: string, value?: any, reason?: any}[] 返回值类型。

5) Promise.race()：用于解决超时问题。

Promise.race([axios('xx'), timeout(1000)]).

自己实现的延时。

6) Generator / Async / Await → 是 Generator 的语法糖

```
var foo = 100

async function main () {
  foo = foo + await Promise.resolve(10)
  console.log('main', foo)
}

main()

foo++
console.log('global', foo)
```

→ 110(后)

→ 101(先)

foo = foo + await Promise.resolve(10)

执行顺序.

step1：确定foo值

step2：确定 await 的值.

step3：相加

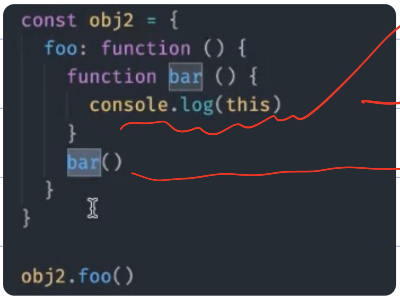
this：指向谁，取决于调用和定义无关.

1) 常规调用) 严格：undefined
 | 非严格：全局

2) new 调用：指向类型对象（空对象）

3) call/apply/bind：手动指定

沿着作用域向上找最近的一个 function，看这个 function 最终是怎样执行的



```
const obj2 = {
  foo: function () {
    function bar () {
      console.log(this)
    }
    bar()
  }
}

obj2.foo()
```

→ 如果是箭头函数，则指向obj2

→ 此时的调用中的this指向全局.

2. ES2018+

① ES2018

1) 展开运算符在对象中应用.

2) 正则表达式的增强.

- 组命名：(?<xxx> \d{3})
- 环视：在匹配的左右看：向后/向前断言 } 肯定 / 否定
- \ 向否定 xxx(?!xxx)

向后肯定 xxx(?=xxx)
向前肯 (?<=A)xxx
向前否 (?<!A)xxx

3) Promise.prototype.finally( )

② ES2019

1) 数组的稳定排序 → 相同值前后顺序排序前后是否打

2) trycatch 后可不接 e

③ ES2020.

A C G J H B D

Router JS:

路由的原理依[illegible]检以变实现、