

# 腾讯2020面经

---

## 一、编程题（请用es6实现编码）：

### 1、二分查找

给定一个 n 个元素有序的（升序）整型数组 nums 和一个目标值 target，写一个函数二分查找 nums 中的 target，如果目标值存在返回下标，否则返回 -1。

```
let arr = [1, 3, 5, 7, 9];
function search(nums, target) {
  let left = 0;
  let right = nums.length - 1;
  while(left <= right) {
    let mid = (right + left) / 2;
    if (nums[mid] == target) {
      return mid;
    } else if (nums[mid] < target) {
      left = mid + 1;
    } else {
      right = mid - 1;
    }
  }
  return -1;
}
console.log(search(arr, 7));
```

### 2、编写一个异步查询等待结果的轮询组件

组件具体输入如下：

1. 查询执行方法
  2. 轮询间隔
  3. 超时时长
  4. 最多轮询次数
- 组件输出：
5. 返回一个promise对象
  6. 查询方法执行无异常，则轮询结束，返回查询结果
- 解答：

// vue代码

```

data() {
  return {
    data: {}, // 请求结果
    remainTime: 0, // 剩余限制时长
    start: 0, // 开始时间
    end: 0, // 请求得到数据的时间
  }
}

mounted () {
  // 请求之前计时 start
  this.start = Math.floor(new Date().getTime()/1000);
  this.query(50, 2, 10);
},
methods: {
  // 只要轮询次数和最大限制时长任何一个达到阈值 就 停止轮询
  query(num, interval, limitTime=60) { // 参数num: 轮询次数 interval: 定时器时间(s) limitTime: 最大限制时长(s)
    let timer;
    if (num > 1) {
      return new Promise((resolve, reject) => {
        let postParams = {
          id: this.$route.params.id
        }
        Api.getArticleOne(postParams).then(res => {
          this.data = res.data.data.content; // 这个只是接口返回的数据结构
          this.end = Math.floor(new Date().getTime()/1000);
        }, err => {
          reject(err);
        }).catch(err => {
          console.log(err)
        });
        if (this.end !== 0) {
          // 如果拿到数据了, 就用当前的 剩余限制时长 - (结束时间 - 开始时间)
          this.remainTime = limitTime - (this.end - this.start);
        } else {
          // 如果还没拿到接口数据 这个时候 end就为0 需要继续 轮询 这个时候的剩余时间
          // 还是 传过来的 限制时长
          this.remainTime = limitTime;
        }

        if (this.remainTime <= 0) { // 如果剩余时间小于等于 0 或者 循环次数 为 1 就结束定
          定时器
          clearTimeout(timer);
          return;
        } else {
          // 轮询一次 num就减少一次
          num--;
          this.remainTime = this.remainTime - interval; // 当前剩余时长 = 当前剩余时长 -
          定时器
        }
      })
    }
  }
}

```

```

        timer = setTimeout(() => { this.query(num, interval, this.remainTime)}, interval *
1000);
    }
    }).catch(err => {
        console.log(err)
    })
    } else {
        clearTimeout(timer);
        return;
    }
    },
}
}

```

### 3、逻辑题

共有60块砖，60人搬，男搬5，女搬3，两个小孩搬1块，一次搬完，需要小孩、男人、女人各多少人，有几种组合方案？

解答：

```

function solution() {
    let x, y, z;
    for(x =1; x < 12; x++) {
        for(y = 1; y < 20; y++) {
            z = 60 - x - y;
            if (z%2 == 0) {
                if (5*x + y*3 + z/2 == 60) {
                    console.log(x, y, z, '搬砖组合'); // 5 3 52
                }
            }
        }
    }
}
solution();

```

答案：只有一种方案：男人：5；女人：3；小孩：52

## 二、问答题

### 1-5题

1. 请写出下面代码输出结果以及原因

```

var myname = "小明";
function showName(){
    console.log(myname); // undefined
    if(0){ var myname = "小红" }
}

```

```
    console.log(myname); // undefined
  }
  showName();
```

## 2. 请写出下面代码输出结果以及原因

```
function letTest() {
  let x = 1;
  if (true) {
    let x = 2;
    console.log(x); // 2
  }
  console.log(x); // 1
}
letTest();
```

## 3. 请写出下面代码输出结果以及原因？并且用箭头函数实现

```
function bar() {
  console.log(myName)
}
function foo() {
  var myName = "腾讯1"
  bar()
}
var myName = "腾讯2"
foo(); // 腾讯2
```

```
箭头函数: var foo = () => () => {
  console.log(myName);
}
```

## 4. 请写出下面代码输出结果以及原因

```
var myObj = {
  name : "腾讯1",
  showThis: function(){
    console.log(this);
    var self = this;
    function bar(){
      self.name = "腾讯2";
    }
    bar()
  }
}
myObj.showThis(); // myObject对象
console.log(myObj.name); // 腾讯2
console.log(window.name); // undefined
```

## 5. 请写出以下this指向情况:

// 情况1

```
function foo() {  
  console.log(this.a) //1  
}
```

var a = 1

foo();

this指向window全局

// 情况2

```
function fn(){  
  console.log(this);  
}
```

var obj = {fn: fn};

obj.fn(); // this => obj this指向obj对象

// 情况3

```
function CreateJsPerson(name,age){
```

// this是当前类的一个实例p1

this.name=name; // => p1.name = name

this.age=age; // => p1.age = age

```
}
```

var p1=new CreateJsPerson("尹华芝",48);

// 情况4

```
function add(c, d){  
  return this.a + this.b + c + d;  
}
```

var o = {a: 1, b: 3};

add.call(o, 5, 7); // 1 + 3 + 5 + 7 = 16 this指向o对象

add.apply(o, [10, 20]); // 1 + 3 + 10 + 20 = 34 this指向o对象

// 情况5

<button id="btn1">箭头函数this</button>

<script type="text/javascript">

let btn1 = document.getElementById('btn1');

let obj = {

name: 'kobe',

age: 39,

getName: function () {

btn1.onclick = () => {

console.log(this); //obj

};

}

};

obj.getName();

this指向obj, 因为箭头函数的this是在外层函数定义的时候就指定了

## 6. 说一说你最近了解的你觉得在前端比较新的技术，并写写你的见解

(开放性题)

node V8引擎，异步I/O，事件机制

Typescript 对变量类型的指定，解除javascript的弱类型的诟病

服务端渲染 (vue-ssr、nuxt、next)：利于seo，更快时间到达，需要服务器支持，加重服务器负载。

## 7. webpack相关

webpac默认的入口文件是index.js

默认输出目录是dist

如何修改webpack的默认输出目录。需要用到webpack命令是webpack

Webpack 常见名词解释，请解释下面名词：

**entry** 项目入口

**module** 模块，对于webpack来说，所有的资源(.js、.css、.png)都是module

**chunk** 打包过程中被操作的模块文件叫做chunk

**bundle** 最后打包后的文件，最终输出的chunk在用户端，被称之为bundle；一般一个chunk对应一个bundle，只有在配置了sourcemap时，才会出现一个chunk对应多个bundle的情况；

**loader** 它就是一个转换器，loader让webpack能够处理不同的文件。loader可以将所有类型的文件转换为webpack能够处理的有效模块，然后利用webpack的打包能力，对他们进行处理。

**plugins** 它就是一个扩展器，它丰富了wepack本身，针对是loader结束后，webpack打包的整个过程，它并不直接操作文件，而是基于事件机制工作，会监听webpack打包过程中的某些节点，执行广泛的任务。

## 8. Node 如何进行缓存

强制缓存和协商缓存,跟浏览器缓存差不多吧

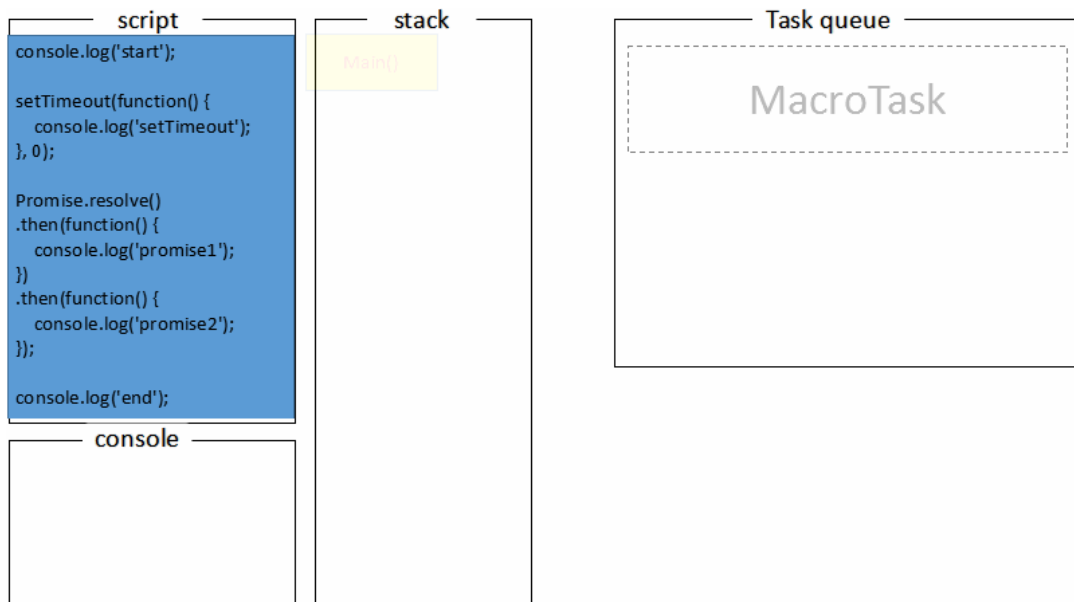
## 9. 浏览器的js循环机制和nodejs循环机制的差别？

[图片上传失败...(image-22e8bf-1586703310728)]

宏任务：setTimeout, setInterval,

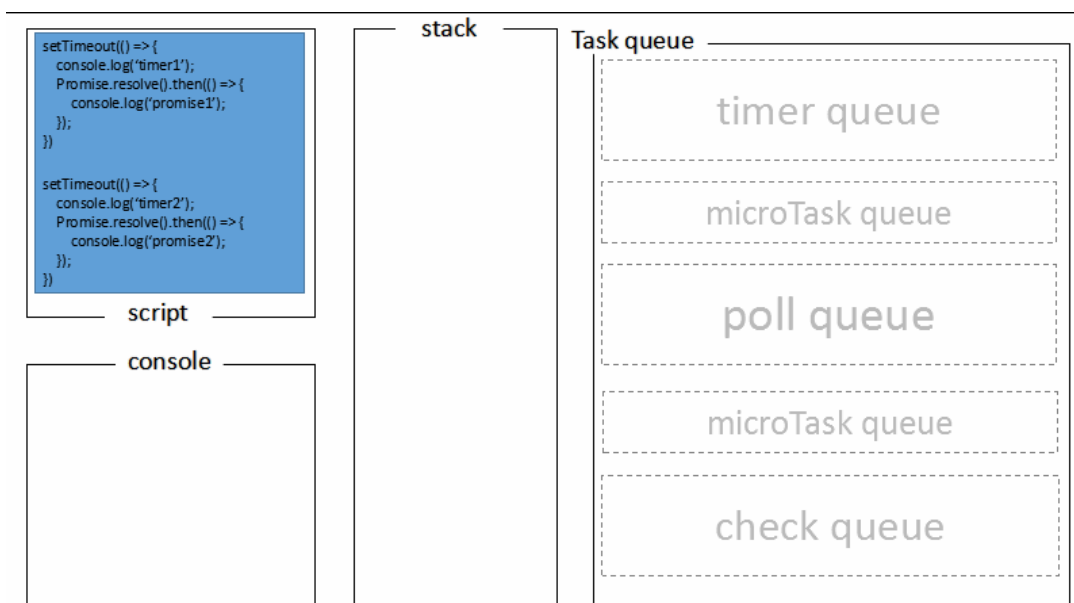
微任务：promise的回调

js的事件循环机制比较简单



先执行主线程代码，执行完毕后，清空微任务队列，然后取出一个宏任务，然后清空微任务队列，如此循环

Node的事件循环比较复杂



Node的事件循环分为六个阶段

(1)timers计时器 执行setTimeout、setInterval的回调函数

(2)I/O callbacks 执行I/O callback被延迟到下一阶段执行；

(3)idle, prepare 队列的移动，仅内部使用

(4)poll 轮询阶段 这个阶段是用来执行和 IO 操作有关的回调的，Node会向操作系统询问是否有新的 IO 事件已经触发，然后会执行响应的事件回调。几乎所有除了 定时器事件、setImmediate() 和 close callbacks 之外操作都会在这个阶段执行。

(5)check 这个阶段会执行 setImmediate() 设置的任务

(6)close 执行close事件的callback，例如socket.on("close",func) 如果一个 socket 或 handle(句柄) 突然被关闭了，例如通过 socket.destroy() 关闭了，close事件将会在这个阶段发出。

## 10. VUE响应式原理解释

**监听器 Observer**：用来劫持并通过Object.defineProperty监听所有属性（转变成 setter/getter形式），如果属性发生变化，就通知订阅者。

**订阅器 Dep**：用来收集订阅者，对 **监听器 Observer** 和 **订阅者 Watcher** 进行统一管理。

**订阅者 Watcher**： **监听器Observer** 和 **解析器Compile** 之间通信的桥梁；如果收到属性的变化通知，就会执行相应的方法，从而更新视图。

**解析器 Compile**：可以解析每个节点的相关指令，对模板数据和订阅器进行初始化。

主要做的事情是：

1. 在自身实例化时往属性 **订阅器(dep)** 里面添加自己。
2. 自身有一个update()方法。
3. 待属性变动dep.notice()通知时，能调用自身的update()方法，并触发 **解析器(Compile)** 中绑定的回调。

**总结**：vue.js 是采用数据劫持结合发布者-订阅者模式的方式，通过Object.defineProperty()来劫持各个属性的setter，getter，在数据变动时发布消息给订阅者，触发相应的监听回调。

### Vue3.0的Proxy相比于defineProperty的优势

Object.defineProperty() 的问题主要有三个：

- 不能监听数组的变化
- 必须遍历对象的每个属性
- 必须深层遍历嵌套的对象

Proxy 在 ES2015 规范中被正式加入，它有以下几个特点：

**针对对象**：针对整个对象，而不是对象的某个属性，所以也就不需要对 keys 进行遍历。这解决了上述 Object.defineProperty() 第二个问题。

**支持数组**：Proxy 不需要对数组的方法进行重载，省去了众多 hack，减少代码量等于减少了维护成本，而且标准的就是最好的。



## 11. HTTP/2、https 有什么新特性

### HTTP/2:

HTTP/2 更简单,高效,强大。

它在传输层解决了以前我们HTTP1.x中一直存在的问题。使用它可以优化我们的应用。HTTP/2的首要目标是通过完全的请求，响应多路复用，头部的压缩头部域来减小头部的体积，添加了请求优先级，服务端推送。为了支持这些特性,他需要大量的协议增加头部字段来支持，例如新的流量控制，差错处理，升级机制.而这些是每个web开发者都应该在他们的应用中用到的。

HTTP/2并没有在应用中改变HTTP的语义，而是通过在客户端和服务端传输的数据格式(frame)和传输.它通过在新的二进制帧层控制整个过程以及隐藏复杂性，而这不需要改变原来有的东西就可以实现。

[详情参考=>](#)

### https:

一、HTTPS协议需要到证书颁发机构CA申请证书，HTTP不用申请证书；

二、HTTP是超文本传输协议，属于应用层信息传输，HTTPS 则是具有SSL加密传安全性传输协议，对数据的传输进行加密，相当于HTTP的升级版；

三、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。

四、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。

## 12. TCP/ip5层模型，并且解释每层的作用

TCP/IP“五层模型”分为：物理层、互联网层、网络层(IP层)、传输层(TCP/UDP层)、应用层。

各层网络协议

应用层(Application):

(application gateway)

Telnet: 远程登录

分应用程序)

FTP (File Transfer Protocol) : 文件传输协议

HTTP (Hyper Text Transfer Protocol) : 超文本传输协议

SMTP (Simple Mail Transter Protocol) : 简单邮件传输协议

POP3 (Post Office Ptotocol) : 邮局协议

SNMP (Simple Network Mangement Protocol) : 简单网络管理协议

DNS (Domain Name System) : 域名系统

应用程序网关

(在应用层连接两部

|  |                              |
|--|------------------------------|
| 传输层 (Transport) :                                    | 传输网关                         |
| (transport gateway)                                  |                              |
| TCP (Transmission Control Potocol): 传输控制协议           | (在传输层连接两个网络)                 |
| UDP (User Data Potocol) : 用户数据协议                     |                              |
| 网络层 (Internet) :                                     | 多协议路由器                       |
| (multiprotocol router)                               |                              |
| IP (Internet Protocol) : 网络协议                        | (在异构网络间转发分组)                 |
| ARP (Address Resolution Protocol) : 地址解析协议           |                              |
| RARP (Reverse Address Resolution Protocol) : 逆地址解析协议 |                              |
| ICMP (Internet Control Message Protocol) : 因特网控制消息协议 |                              |
| IGMP (Internet Group Manage Protocol) : 因特网组管理协议     |                              |
| BOOTP (Bootstrap) : 可选安全启动协议                         |                              |
| 互联网层 即 数据链路层 (Data Link) :                           | 网桥                           |
| (bridge) 交换机 (switcher)                              |                              |
| HDLCL (High Data Link Control) : 高级数据链路控制            | (在LAN之间存储-转发数据链路针)           |
| SLIP (Serial Line IP) : 串行线路IP                       |                              |
| PPP (Point-to-Point Protocol) : 点到点协议                |                              |
| 802.2等   |                              |
| 物理层 (Physical) :                                     | 中继器 (repeater) 集线器 (hub)     |
| 无  | (放大或再生弱的信号, 在两个电缆段之间复制每一个比特) |

**应用层** :应用程序间沟通的层, 如简单电子邮件传输 (SMTP)、文件传输协议 (FTP)、网络远程访问协议 (Telnet) 等。

**传输层** : 在此层中, 它提供了节点间的数据传送服务, 如传输控制协议 (TCP)、用户数据报协议 (UDP) 等, TCP和UDP给数据包加入传输数据并把它传输到下一层中, 这一层负责传送数据, 并且确定数据已被送达并接收。

**数据链路层** : O S I 模型的第二层, 它控制网络层与物理层之间的通信。它的主要功能是如何在不可靠的物理线路上进行数据的可靠传递。为了保证传输, 从网络层接收到的数据被分割成特定的可被物理层传输的帧。帧是用来移动数据的结构包, 它不仅包括原始数据, 还包括发送方和接收方的网络地址以及纠错和控制信息。其中的地址确定了帧将发送到何处, 而纠错和控制信息则确保帧无差错到达。

数据链路层的功能独立于网络和它的节点和所采用的物理层类型, 它也不关心是否正在运行 Word、Excel 或使用 Internet。有一些连接设备, 如交换机, 由于它们要对帧解码并使用帧信息将数据发送到正确的接收方, 所以它们是工作在数据链路层的。

**网络层** : O S I 模型的第三层, 其主要功能是将网络地址翻译成对应的物理地址, 并决定如何将数据从发送方路由到接收方。

网络层通过综合考虑发送优先权、网络拥塞程度、服务质量以及可选路由的花费来决定从一个网络中节点A 到另一个网络中节点B 的最佳路径。由于网络层处理路由, 而路由器因为即连接网络各段, 并智能指导数据传送, 属于网络层。在网络中, “路由”是基于编址方案、使用模式以及可达性来指引数据的发送。

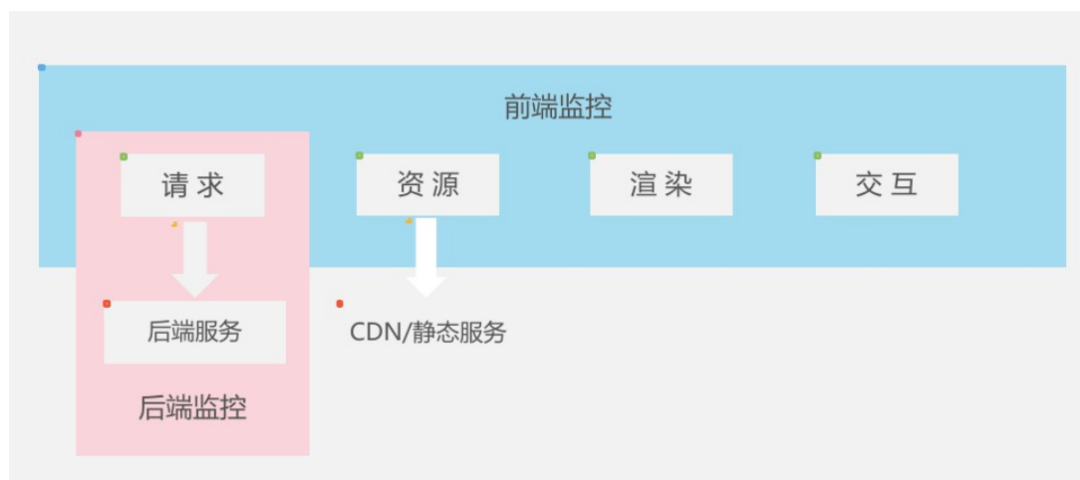
**物理层**：O S I 模型的最低层或第一层，该层包括物理连网媒介，如电缆连线连接器。物理层的协议产生并检测电压以便发送和接收携带数据的信号。在你的桌面P C 上插入网络接口卡，你就建立了计算机连网的基础。换言之，你提供了一个物理层。尽管物理层不提供纠错服务，但它能够设定数据传输速率并监测数据出错率。网络物理问题，如电线断开，将影响物理层。

### 13. 从前端的角度，如何系统性的提升大型 Web 应用的可用性？可从你认为的可用性维度、监控手段和改善措施等方面阐述。

#### 一、前端用户性意义与现状

什么是前端可用性？

前端可用性是从用户的角度出发，检测整个系统的可用性，系统任何一个环节的缺失都会对体验造成影响。



#### 前端可用性现状

1. 页面功能和交互复杂度增加
2. 前端功能测试局限性
3. 各种前端渲染框架的引入
4. 运营线上问题反馈

#### 前端可用性建设意义



### 前端可用性评估指标

- **关键指标**白屏时间6s

### 前端可用性保障系统设计

可用性系统要求：实时性，全面性

#### 1. 数据采集：请求异常、资源异常、渲染异常、交互异常。

请求状态吗异常，请求超时，返回数据格式错误。（AJAX监控）

资源加载失败（CDN监控）

渲染异常（DOM检查）

交互异常（JS错误监控）

#### 2. 监控预警：实时监控、阈值报警。

海量数据存储读取、可视化数据展现、多维度数据查询。

设定合理阈值、邮件短信报警

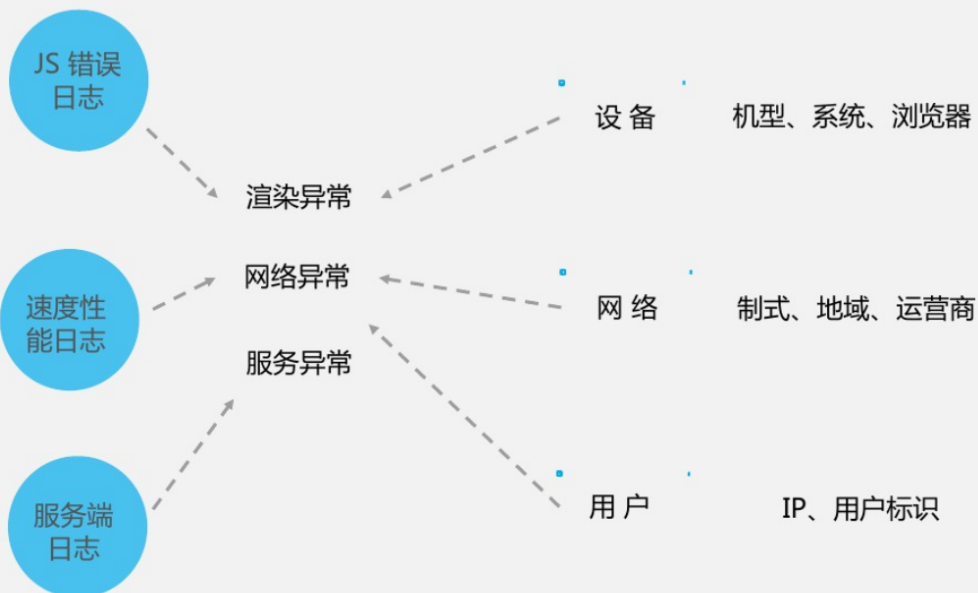
#### 3. 兜底容灾：容错机制、快速降级。

异步渲染机制出错跳转同步页、友好的错误用户提示。

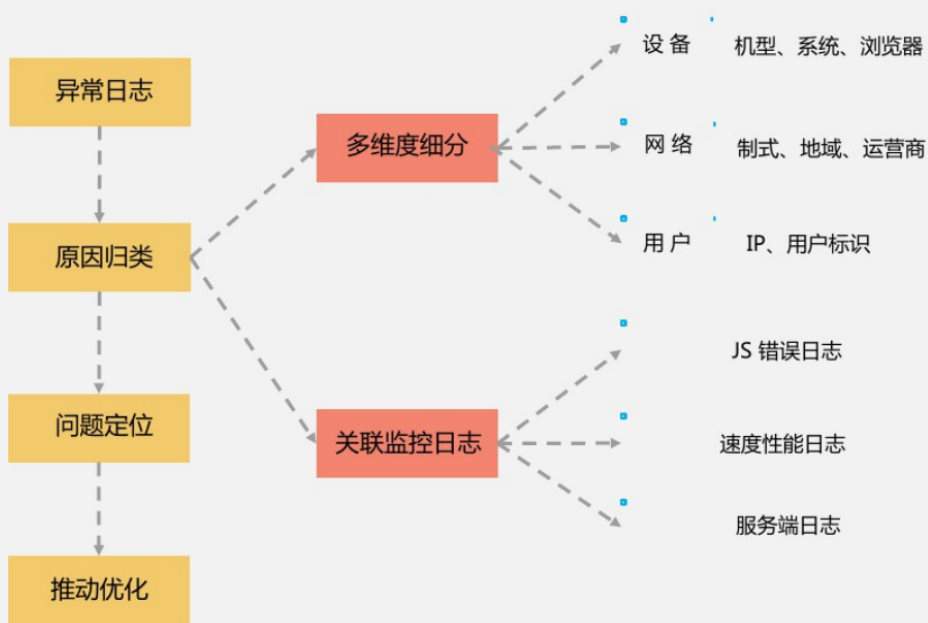
重要机制添加降级开关、迅速（3min内）完成降级

### 前端可用性优化思路

如何优化某产品的前端可用性？



运用统计思维分析数据，还原用户交互过程。



**Good Case1:**  **实时性**


某产品线前端配置上线 -----> **多模块耦合，功能测试局限**


上线过程中发现端上白屏指标数量突增 -----> **兜底监控，端上核心服务指标**


触发报警机制 -----> **实时反馈问题**


迅速进行降级回滚 -----> **快速降级挽回损失**

## Good Case2: 全面性

某小流量实验转全，后端服务监控表现正常  后端监控覆盖不全

白屏展现数量大幅上升  无法立即定位问题

数据分析前端代码有浏览器兼容性问题，  
某些浏览器下发生白屏、样式错乱等问题  结合优化分析思路

迅速修复问题，挽回流量损失  主动反馈线上问题

### 前端可用性展望



### 面试总结

主要就是结合简历上的一个一个详细地问，只要简历上写了就会问到原理性的东西，然后就是问了项目中你负责哪块，前端规划之类的。

然后针对前端重复页面的编写探讨了一下，面试官小姐姐给我解答了，他们后台有自己的组件库，纯的组件，前端要用，就直接引用下载，开始不以为然，到下来自己去查了一下，大概就是npm发包的那种吧。