阿里篇——

1、Typescript 中的 interface 和 type 有什么区别？

① interface 定义数据的形状，具体这个数据结构如何，有哪些属性，如定义一个 object；type 定义数据的类型，指一个数据具体值是什么类型，如 boolean，string。

② interface 可以被 class 继承和实现，也可以继承 class；但 type 不行

③ interface 不能作为交叉、联合类型的产物，但可以作为其组成元素；而 type 就没有什么限制既可以作为组成元素又可作为产物

④ 使用 type 可以自定义范型函数，为己所用；interface 不行

但是在日常的工作中他俩还真没有太大区别，特殊场景单独分析，基本上都能怎么舒服怎么来

2、用 javascript 实现一个 dispatch 函数，可以根据不同的参数打印出不同的字符串，同时支持链式调用：

①当传入字符串"a"时，可以直接打印出字符串"a"。

```
dispatch('a');
> "a"
```

②当调用 println("b") 方法时，可以打印出字符串"b"。

```
dispatch('a').println('b')
> "a"
> "b"
```

③当调用 wait(n) 方法时，可以先等待 n 秒，然后再执行后面的操作。

```
dispatch('a').wait(3).println('b')
> "a"
> 3 秒后
> "b"
```

④当调用 waitFirst(n) 方法时，所有操作延后，先等待 n 秒，。

```
dispatch('a').waitFirst(3).println('b')
> 3 秒后
> "a"
> "b"
```

参考答案：

```
function dispatch() {
  let chain = Promise.resolve();
  let tasks = [];
```

```
  return {
    println(text) {
      tasks.push(() => console.log(text));
      return this;
    },
    wait(seconds) {
      tasks.push(() => new Promise(resolve => setTimeout(resolve, seconds * 1000)));
      return this;
    },
    waitFirst(seconds) {
      chain = chain.then(() => new Promise(resolve => setTimeout(resolve, seconds * 1000)));
      return this;
    },
    exec() {
      chain = tasks.reduce((prev, next) => {
        return prev.then(next);
      }, chain);
      tasks = [];
    }
  };
}
```

主要考察 Promise ，异步流程以及链式调用

3、> 实现一个 AST 解析方法，解析下列输入，输出对应树形结构（区分标签、属性、内容等）

```
/**
 * 实现一个 AST 解析方法，解析下列输入，输出对应树形结构（区分标签、属性、内容等）
 * @param str
 */
function astParser(str){
  // your code are here...
}

// 用例参考
astParser(`
  <div class="widget-body" data-spm-anchor-id="a1z4o.xxss.i3.14803e15bAFF41">
    <span class="ctr-val g-csscut-more" style="display: inline-block;vertical-align: top;width:200px;”><a target="_blank" href=" " title="欢迎应聘蚂蚁金服支付宝前端工程师-杭州、上海、北京、成都">欢迎应聘蚂蚁金服支付宝前端工程师-杭州、上海、北京、成都</a ></span>
  </div>
`);
```

```
// 返回结果参考：{
//        tagName: 'div',
//        classnames: 'widget-body',
//        style: '',
//        atrributes: {
//              'data-spm-anchor-id': 'a1z4o.xxss.i3.14803e15bAFF41'
//        },
//        innerText: '',
//        children: [{
//              tagName: 'span',
//              classnames: 'ctr-val g-csscut-more',
//              style: 'display: inline-block;vertical-align: top;width:200px;',
//              atrributes: {},
//              innerText: '',
//              children: [{
//                    tagName: 'a',
//                    classnames: '',
//                    style: '',
//                    atrributes: {
//                          target: '_blank',
//                          href: 'positionDetail.htm?id=44106',
//                          title: '欢迎应聘蚂蚁金服支付宝前端工程师-杭州、上海、北京、成都',
//                    },
//                    innerText: '欢迎应聘蚂蚁金服支付宝前端工程师-杭州、上海、北京、成都
',
//              }]
//        }]
// }
```

参考答案：

```
function astParser(str) {
      var stack = [];
      var root;
      var tokens = str.split(/(?=<\/?[^>]+>)/g).filter(Boolean);
      for (var i = 0; i < tokens.length; i++) {
            var token = tokens[i];
            if (token[1] !== '/') {
                  var tagMatch = token.match(/<([^ >]+)/);
                  var attrMatch = token.match(/ ([^=>]+)="([^"]+)"/g);
                  var tagName = tagMatch ? tagMatch[1] : '';
                  var attributes = {};
                  var classnames = '';
                  if (attrMatch) {
                        attrMatch.forEach(function(attr) {
```

```
                    var parts = attr.split('=');
                    var attrName = parts[0].trim();
                    var attrValue = parts[1].replace(/"/g, '');
                    if (attrName === 'class') {
                        classnames = attrValue;
                    } else {
                        attributes[attrName] = attrValue;
                    }
                });
            }
            var node = {
                tagName: tagName,
                classnames: classnames,
                attributes: attributes,
                children: [],
                innerText: '',
            };
            if (stack.length) {
                stack[stack.length - 1].children.push(node);
            } else {
                root = node;
            }
            stack.push(node);
        } else if (token[1] === '/') {
            if (stack.length) {
                stack.pop();
            }
        } else {
            stack[stack.length - 1].innerText += token;
        }
    }
    return root;
}
```

这题解法比较多，如果使用现有的库也可以，可以使用以上这种正则方式来直接匹配出不同的类型，也可以使用状态机的方式挨个匹配字符串，主要考察编译原理的解题思路，不需要在短时间内解出

4、给你一个字符串表达式 s ，请你实现一个基本计算器来计算并返回它的值。
注意:不允许使用任何将字符串作为数学表达式计算的内置函数，比如 eval() 。
示例 1：
输入：s = "1 + 1"
输出：2
示例 2：

输入：s = " 2-1 + 2 "
输出：3
示例 3：

输入：s = "(1+(4+5+2)-3)+(6+8)"
输出：23
提示：
1 <= s.length <= 3 * 105
s 由数字、'+'、'-'、'('、')'、和 ' ' 组成
s 表示一个有效的表达式
'+' 不能用作一元运算(例如， "+1" 和 "+(2 + 3)" 无效)
'-' 可以用作一元运算(即 "-1" 和 "-(2 + 3)" 是有效的)
输入中不存在两个连续的操作符
每个数字和运行的计算将适合于一个有符号的 32 位 整数

参考答案

```
function calculate(s) {
    let stack = [];
    let operand = 0;
    let result = 0;
    let sign = 1;

    for (let i = 0; i < s.length; i++) {
        let ch = s.charAt(i);

        if (ch >= '0' && ch <= '9') {
            operand = (operand * 10) + (ch - '0');
        } else if (ch === '+') {
            result += sign * operand;
            sign = 1;
            operand = 0;
        } else if (ch === '-') {
            result += sign * operand;
            sign = -1;
            operand = 0;
        } else if (ch === '(') {
            stack.push(result);
            stack.push(sign);
            sign = 1;
            result = 0;
        } else if (ch === ')') {
            result += sign * operand;
            operand = 0;
            result *= stack.pop();
```

```
                result += stack.pop();
            }
        }
        return result + (sign * operand);
};
```

可以使用栈或者递归来实现，也可以使用状态机来做，进一步可以考察时间空间复杂度

5、

```
/**
 * 大数相乘，限制：不可用 BigInt
 *
 * 例如：
 * 输入
 * a = '1111111111111111111'
 * b = '22222222222222222222'
 * 返回
 * '24691358024691358024197530864197530864 2'
 *
 * @param {string} a
 * @param {string} b
 * @return {string}
 */
const multiplied = (a, b) =>{
    //这里写 javascript 代码实现

  }
```

参考答案

```
const multiplied = (a, b) => {
    if(a === '0' || b === '0') return '0';
    let m = a.length, n = b.length;
    let res = Array(m + n).fill(0);
    for(let i = m - 1; i >= 0; i--) {
        for(let j = n - 1; j >= 0; j--) {
            let mul = (a[i] - '0') * (b[j] - '0');
            let p1 = i + j, p2 = i + j + 1;
            let sum = mul + res[p2];
            res[p2] = sum % 10;
            res[p1] += Math.floor(sum / 10);
        }
    }
    while(res[0] === 0) {
        res.shift();
    }
    return res.join('');
```

}

考察候选人考虑问题的方便和基础代码能力

6、// JS 实现一个带并发限制的异步调度器 Scheduler，
// 保证同时运行的任务最多有两个。
// 完善代码中 Scheduler 类，
// 使得以下程序能正确输出

```
class Scheduler {
  constructor() {
    this.count = 2
    this.queue = []
    this.run = []
  }

  add(task) {
                    // ...
  }
}


const timeout = (time) => new Promise(resolve => {
  setTimeout(resolve, time)
})

const scheduler = new Scheduler()
const addTask = (time, order) => {
  scheduler.add(() => timeout(time)).then(() => console.log(order))
}

addTask(1000, '1')
addTask(500, '2')
addTask(300, '3')
addTask(400, '4')
// output: 2 3 1 4

// 一开始，1、2 两个任务进入队列
// 500ms 时，2 完成，输出 2，任务 3 进队
// 800ms 时，3 完成，输出 3，任务 4 进队
// 1000ms 时，1 完成，输出 1
// 1200ms 时，4 完成，输出 4
```

参考答案

```
class Scheduler {
  constructor() {
    this.count = 2;
    this.queue = [];
    this.run = [];
  }

  add(task) {
    return new Promise((resolve, reject) => {
      // 任务对象，包含执行的任务和 resolve 方法
      const job = { task, resolve };
      if (this.run.length < this.count) {
        // 如果运行中的任务数量小于限制，直接运行任务
        this.runJob(job);
      } else {
        // 否则将任务加入队列等待
        this.queue.push(job);
      }
    });
  }

  runJob(job) {
    this.run.push(job);
    job.task().then(() => {
      // 任务完成后调用 resolve，并从运行中的任务数组中移除
      job.resolve();
      this.completeJob(job);
    });
  }

  completeJob(job) {
    const index = this.run.indexOf(job);
    if (index > -1) {
      this.run.splice(index, 1);
    }
    // 如果队列中还有任务，继续运行
    if (this.queue.length > 0) {
      this.runJob(this.queue.shift());
    }
  }
}
```

考察基础编程能力

7、JS 数组的去重方式有哪些？

方法一

```
let arr = [1, 2, 2, 3]
console.log([...new Set(arr)])
```

方法二

```
let arr = [1, 2, 2, 3]
function unique(arr) {
    let tmpArr = []
    for (let i = 0; i < arr.length; i++) {
        if (!tmpArr.includes(arr[i])) {
            tmpArr.push(arr[i])
        }
    }
    return tmpArr
}
console.log(unique(arr))
```