携程前端面试

携程一面

携程面试最大的特点是：半聊天半技术的面试方式，面试官是做前端基础架构的，所以对于基础非常重视，不搞花里胡哨的东西。

（1）常规自我介绍

（2）前端如何进行权限管理的?

接口级权限：

通过登录判断用户是否为管理员或是普通用户，根据各自的角色向后台获取页面url，前端根据url进行页面渲染，有权限的显示，无权限的不显示。

页面级权限：

页面及权限主要靠 `vue-router` 来实现。基本思路是为全局注册一个“前置守卫”钩子函数 `router.beforeEach` 。

```
const router = new VueRouter({ ... })

router.beforeEach((to, from, next) => {
  // 这里检查权限并进行跳转
  next()
})

```

每个守卫方法接收三个参数：

- to: `Route` 即将要进入的目标
- from: `Route` 当前导航正要离开的路由
- next: `Route` 接下来要进行执行的任务 方法的调用参数。
- next(): 进行管道中的下一个钩子。如果全部钩子执行完了，则导航的状态就是 `confirmed` （确认的）。
- next(false): 中断当前的导航。如果浏览器的 `URL` 改变了（可能是用户手动或者浏览器后退按钮），那么 `URL` 地址会重置到 `from` 路由对应的地址。

- `next('/')` 或者 `next({ path: '/' }):` 跳转到一个不同的地址。当前的导航被中断，然后进行一个新的导航。你可以向 next 传递任意位置对象，且允许设置诸如 `replace: true`、`name: 'home'` 之类的选项以及任何用在 `router-link` 的 `to prop` 或 `router.push` 中的选项。
- next(error): (2.4.0+) 如果传入 `next` 的参数是一个 `Error` 实例，则导航会被终止且该错误会被传递给 `router.onError()` 注册过的回调。 *确保要调用 `next` 方法，否则钩子就不会被 `resolved`。*

（3）知道如何使用npm进行插件发布嘛？

```
npm init //初始化项目
npm login //登录npm用户
//创建js文件
npm publish .//发布上传包文件
npm install 包名 //下载安装包
```

（4）如何实现按需加载和路由懒加载？

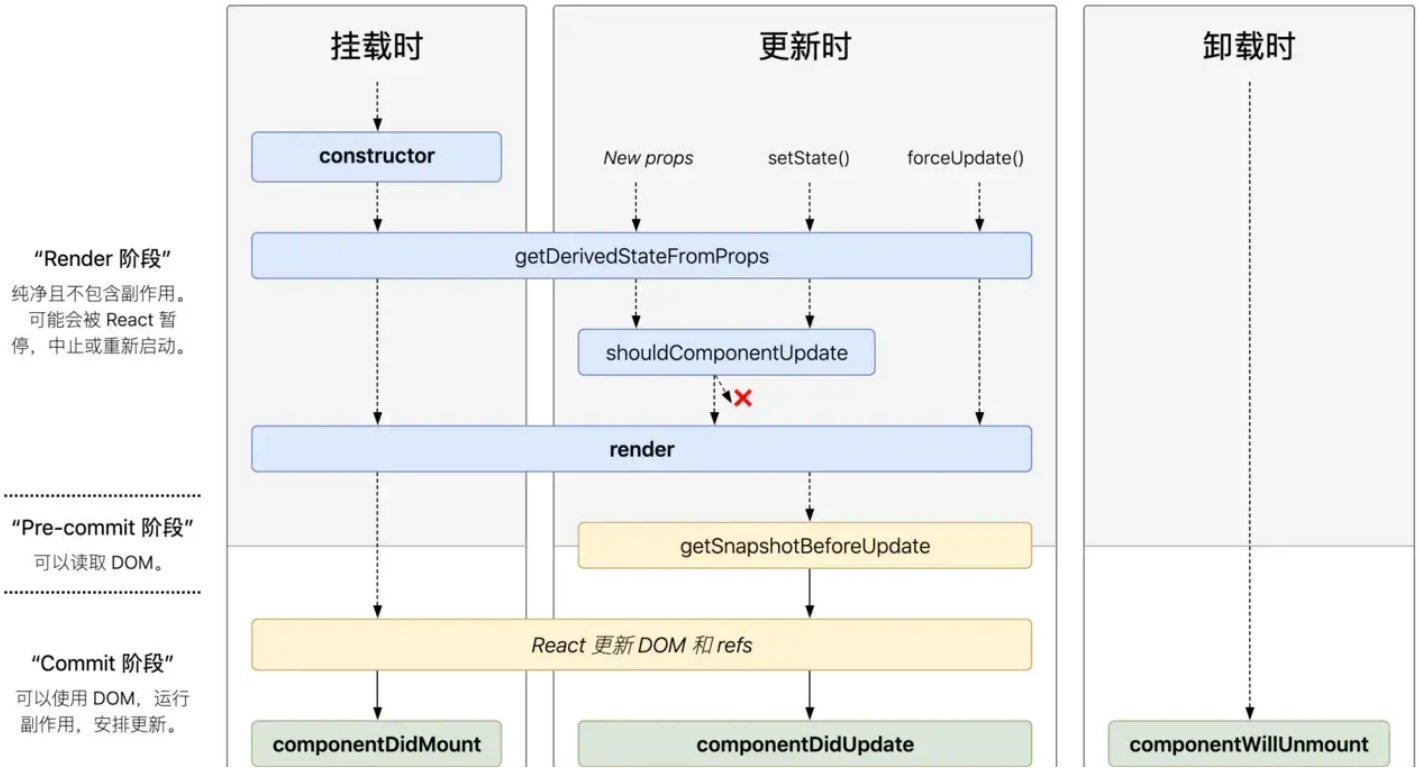
vue的按需加载：vue-router配置路由，使用Vue异步组件技术，实现异步加载。

懒加载：当打包构建应用时，JavaScript 包会变得非常大，影响页面加载。如果我们能把不同路由对应的组件分割成不同的代码块，然后当路由被访问的时候才加载对应组件，这样就更加高效了。结合 Vue 的[异步组件]和 Webpack 的[代码分割功能]，轻松实现路由组件的懒加载。

（5）git进行项目管理有哪些命令？

命令	说明
git init	初始化仓库
git clone	拷贝一份远程仓库，也就是下载一个项目。
git add	添加文件到仓库
git status	查看仓库当前的状态，显示有变更的文件。
git diff	比较文件的不同，即暂存区和工作区的差异。
git commit	提交暂存区到本地仓库
git reset	回退版本。
git rm	删除工作区文件
git mv	移动或重命名工作区文件。
git log	查看历史提交记录
git blame	以列表形式查看指定文件的历史修改记录
git remote	远程仓库操作
git fetch	从远程获取代码库
git pull	下载远程代码并合并
git push	上传远程代码并合并

（6）React的生命周期有哪些？初始渲染数据放在哪个生命周期中？



挂载前后、更新前后、卸载前后

初始渲染数据只用一次的，放在挂载前。

（7）常用的跨域处理？

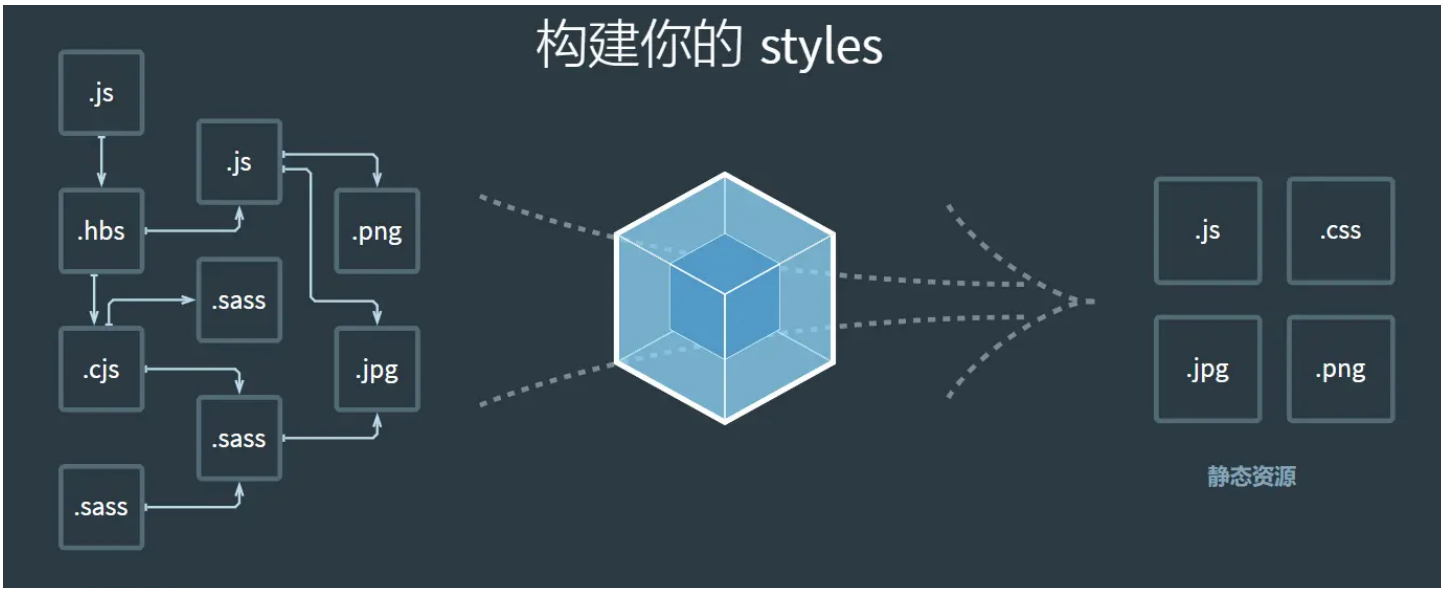
jsonp、proxy、nginx反向代理、后台设置cors、script导入外部文件

（8）常用的异步处理方式？

Promise、async await、回调

（9）webpack配置和原理？

webpack是前端用来构建文件的管理工具，可以将各种插件创建的文件、各种版本资源的文件，打包成各种版本浏览器兼容的代码、资源文件。webpack是依赖于npm创建的插件工具 。



具体流程就是： 首先各种模块文件导入到webpack打包配置文件中 将模块文件解析成chunk代码块，对各个模块间的关系进行静态分析 根据模块的依赖关系进行各种文件打包成各种版本浏览器可识别的文件 配置浏览器入口文件bundle

Entry：入口文件，指示webpack以哪个文件为入口起点开始打包，分析构建内部依赖图。 Output：输出文件，指示webpack打包生成的bundles输出到哪里，以及如何命名 Loader：预解析器，让webpack能够去处理那些非js文件 Plugins：插件，用于处理更加丰富的任务，从打包优化到压缩，一直到重新定义环境中的变量等。 Model：模式，由于指示webpack使用响应环境模式的配置信息。

(10)常见的设计模式?

3.携程二面

（1）TCP和UDP的区别?

TCP（Transmission Control Protocol，传输控制协议）是基于连接的协议，也就是说，在正式收发数据前，必须和对方建立可靠的连接。一个TCP连接必须要经过三次“对话”才能建立起来。

UDP（User Data Protocol，用户数据报协议）是与TCP相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去！ UDP适用于一次只传送少量数据、对可靠性要求不高的应用环境 。

（2）关于HTTP2.0你知道多少?

HTTP/2引入了“服务端推（server push）”的概念，它允许服务端在客户端需要数据之前就主动地将数据发送到客户端缓存中，从而提高性能。 HTTP/2提供更多的加密支持HTTP/2使用多路技术，允许多个消息在一个连接上同时交差。它增加了头压缩（header compression），因此即使非常小的请求，其请求和响应的header都只会占用很小比例的带宽 。

（3）https为什么是安全的？非对称加密解密的过程?

http是超文本传输协议，信息是明文进行传输的。https是具有安全性的ssl加密传输协议，为浏览器和服务器之间的通信加密，确保数据传输的安全。HTTP协议通常承载于TCP协议之上，在HTTP和TCP之间添加一个安全协议层（SSL或TSL），这个时候，就成了我们常说的HTTPS。

HTTPS选择了握手时交换密钥的方案。

总的来说，握手过程中，服务器会发出一张证书（带着公钥），客户端用公钥加密了一段较短的数据S，并返回给服务器。服务器用私钥解开，拿到S。此时，握手步骤完成，S成为了一个被安全传输到对方手中的对称加密密钥。此后，服务器与我的请求响应，只需要用S作为密钥进行一次对称的加密就好。

（4）简单的用户登录逻辑?

`token` 有三种状态

- 无值
- 有值，有效
- 有值，过期

`token` 的作用：和后端进行交互时，作为通行证。

`http` 拦截分为两种：

- 请求拦截
- 响应拦截

【1】请求拦截：如果本地有 `token` ,在发送请求前请求头 `headers` 添加 `token` 字段。 【2】响应拦截：当前token失效了，但是token依然保存在本地。这时候你去访问需要登录权限的接口时，实际上应该让用户重新登录。这时候就需要结合 http 拦截器 + 后端接口返回的http 状态码来判断。

（5）讲一讲浏览器的事件循环Event Loop?

事件循环主要涉及的概念是 事件队列、微任务、宏任务

在JavaScript中，任务被分为Task（又称为MacroTask,宏任务）和MicroTask（微任务）两种。它们分别包含以下内容：

宏任务：setTimeout setInterval setImmediate（ie下生效） MessageChannel（消息通道）

微任务：Promise.then MutationObserver（监听dom节点更新完毕） process.nextTick()（node的文法，比Promise.then执行的快）

代码从上到下执行，会先执行同步的代码，再执行微任务，等待宏任务有没有到时间，时间到了的宏任务放到宏任务队列，微任务执行完毕后，会从宏任务队列中取出一个宏任务去放到当前的浏览器的执行环境中执行，当前执行环境都执行完毕后，会先去清空微任务，下面我们来看一段代码.

```
1 setTimeout(()=>{
2     console.log('timeout1');
```

```
    Promise.resolve().then(data=>{
        console.log('then1')
    })
},0)
Promise.resolve().then(data=>{
    console.log('then2')
    setTimeout(()=>{
        console.log('timeout2');
    },0)
})
//then2 timeout1 then1 timeout2
```

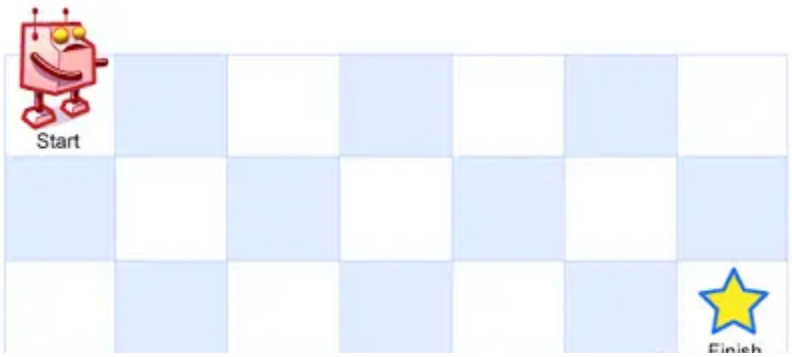
同样是异步，为什么要有宏任务呢?

因为宏任务涉及到调用浏览器的io接口进行运行，所以宏任务总是在微任务结束后进行，

（6）CSRF跨站点请求伪造?

CSRF：跨站点请求伪造，其原理是攻击者构造网站后台某个功能接口的请求地址，有道用户去点击或者用特殊方法让该请求地址自动加载。用户在登录状态下这个请求被服务端接收后会被误以为是用户的合法的操作。对于GET形式的接口地址可轻易被攻击，对于POST形式的接口地址也不是百分百安全，攻击者可以诱导用户进入待FORM表单可有的POST方式提交参数的页面。

（7）算法题：给定一个m×n网格，左上角起点标记为"Start"，每次移动一步，求走到Finish总共有多少条不同的路径？（动态规划）



例如：上图是一个7×3的网格，求有多少中可能的路径?

示例1：

```
输入: m = 3, n = 2
输出: 3
解释:
从左上角开始，总共有 3 条路径可以到达右下角。

向右 -> 向右 -> 向下
```

```
向右 -> 向下 -> 向右
向下 -> 向右 -> 向右
```

示例2：

```
输入: m = 7, n = 3
输出: 28
```

- `1 <= m, n <= 100`
- 题目数据保证答案小于等于 `2 * 10 ^ 9`

根据移动规则，我们可以确定，在 [i, j] 位置的点，只能通过 [i-1, j] 以及 [i, j-1] 到达，由以上分析，我们可以写出 `solution` 代码。

```cpp
class Solution {
public:
    // 使用动态规划解题,能够到达每个格子，只能是从他的上方或者左侧到达
    int uniquePaths(int m, int n)
    {
        // 建立dp
        vector<vector<int>>dp(m, vector<int>(n, 0));
        // 填充第一行
        for(int j=0; j<n; ++j)
            dp[0][j] = 1;
        // 填充第一列
        for(int i=0; i<m; ++i)
            dp[i][0] = 1;
        // 填充剩余空间
        for(int i=1; i<m; ++i)
        {
            for(int j=1; j<n; ++j)
                dp[i][j] = dp[i-1][j] + dp[i][j-1];
        }

        return dp[m-1][n-1];
    }
};
```

对于该解决方案，其时间复杂度与空间复杂度为：

- 时间复杂度： O(m * n)
- 空间复杂度： O(m * n)

我们知道，通常对于DP题目的初始解法，是可以对其空间复杂度进行优化的，比如这道题目，我们刚开始用了二维数组来存储，所以消耗空间为 O(m * n) 但我们完全可以压缩为一维数组存储，下面我们就来尝试实现一下。

依然记得，规则：路径数 （i,j) = （i-1,j) + （i, j-1)

```java
class Solution {
    public int uniquePaths(int m, int n) {
        int[] result = new int[m];
        // 初始化首行信息
        for(int i = 0; i < m; i++) result[i] = 1;
        for(int i = 1; i < n; i++){
            for(int j = 1; j < m; j++){
                // 更新数组，当前位置 = 更新后的左侧 + 更新前的上一位
                result[j] += result[j-1];
            }
        }
        return result[m-1];
    }
}
```

最后附上JS代码：

```javascript
//动态规划实现最小路径和

//这里的m和n可以手动输入，利用prompt语句。这里指定一下值，但下面的程序用的都是m和n，如果需要修改m和n的值只需要修改定义这里的就行。
var m = 4;
var n = 5;
/*不太会定义结构体，所以定义了三个二维数组,基本算法思想是一样的*/
//up数组存储每个坐标的向上的路径，right存储每个坐标向下的路径，v数组利用动态规划存储该坐标的值到右上终点的最小路径和
var up = new Array();
for(var i=1;i<=m;i++){
    up[i] = new Array();
    for(var j=1;j<=n;j++){
        up[i][j] = 0;
    }
}

var right = new Array();
for(var i=1;i<=m;i++){
    right[i] = new Array();
    for(var j=1;j<=n;j++){
```

```
        right[i][j] = 0;
    }
}

var v = new Array();
for(var i=1;i<=m;i++){
    v[i] = new Array();
    for(var j=1;j<=n;j++){
        v[i][j] = Number(0);
    }
}

//从[1,1]到[m,n]依次输入所在坐标的up和right值
for(var i=1;i<=m;i++){
    for(var j=1;j<=n;j++){
        up[i][j] = prompt("Please input 'up'  a["+i+"]["+j+"]");
        right[i][j] = prompt("Please input 'right' a["+i+"]["+j+"]");
    }
}

//compute
v[m][n] = Number(0);
for(i=m;i>0;i--)
{
    for(j=n;j>0;j--)
    {
        if( m==i && j!=n)
        {
            v[i][j] = Number(right[i][j]) + Number(v[i][j+1]);
        }
        else if( n==j && i!=m)
        {
            v[i][j] = Number(up[i][j]) + Number(v[i+1][j]);
        }
        else if(i!=m && j!=n)
        {
            var n1 = Number(up[i][j]) + Number(v[i+1][j]);
            var n2 = Number(right[i][j])+Number(v[i][j+1]);
            //要求的是最短路径,所以三元表达式如下:
            v[i][j] = Number(n1>n2?n2:n1);
        }
    }
}
```

4.携程三面?

携程三面是HR面，主要是询问的是自己在项目中扮演的一些角色，对于前端岗位的一些理解，还有自己手上的一些offer情况。

5.总结

携程面试整体偏基础，侧重对原理进行探讨。以上是面经总结，如有错误欢迎指出。