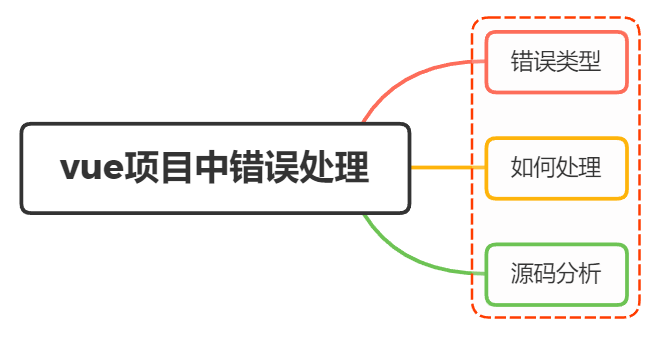
# 面试官：你是怎么处理vue项目中的错误的？



## 一、错误类型

任何一个框架，对于错误的处理都是一种必备的能力

在Vue 中，则是定义了一套对应的错误处理规则给到使用者，且在源代码级别，对部分必要的过程做了一定的错误处理。

主要的错误来源包括：

* 后端接口错误
* 代码中本身逻辑错误

## 二、如何处理

### 后端接口错误

通过axios的interceptor实现网络请求的response先进行一层拦截

apiClient.interceptors.response.use(  
 response => {  
 return response;  
 },  
 error => {  
 if (error.response.status == 401) {  
 router.push({ name: "Login" });  
 } else {  
 message.error("出错了");  
 return Promise.reject(error);  
 }  
 }  
);

### 代码逻辑问题

#### 全局设置错误处理

设置全局错误处理函数

Vue.config.errorHandler = function (err, vm, info) {  
 // handle error  
 // `info` 是 Vue 特定的错误信息，比如错误所在的生命周期钩子  
 // 只在 2.2.0+ 可用  
}

errorHandler指定组件的渲染和观察期间未捕获错误的处理函数。这个处理函数被调用时，可获取错误信息和 Vue 实例

不过值得注意的是，在不同Vue 版本中，该全局 API 作用的范围会有所不同：

从 2.2.0 起，这个钩子也会捕获组件生命周期钩子里的错误。同样的，当这个钩子是 undefined 时，被捕获的错误会通过 console.error 输出而避免应用崩

从 2.4.0 起，这个钩子也会捕获 Vue 自定义事件处理函数内部的错误了

从 2.6.0 起，这个钩子也会捕获 v-on DOM 监听器内部抛出的错误。另外，如果任何被覆盖的钩子或处理函数返回一个 Promise 链 (例如 async 函数)，则来自其 Promise 链的错误也会被处理

#### 生命周期钩子

errorCaptured是 2.5.0 新增的一个生命钩子函数，当捕获到一个来自子孙组件的错误时被调用

基本类型

(err: Error, vm: Component, info: string) => ?boolean

此钩子会收到三个参数：错误对象、发生错误的组件实例以及一个包含错误来源信息的字符串。此钩子可以返回 false 以阻止该错误继续向上传播

参考官网，错误传播规则如下：

* 默认情况下，如果全局的 config.errorHandler 被定义，所有的错误仍会发送它，因此这些错误仍然会向单一的分析服务的地方进行汇报
* 如果一个组件的继承或父级从属链路中存在多个 errorCaptured 钩子，则它们将会被相同的错误逐个唤起。
* 如果此 errorCaptured 钩子自身抛出了一个错误，则这个新错误和原本被捕获的错误都会发送给全局的 config.errorHandler
* 一个 errorCaptured 钩子能够返回 false 以阻止错误继续向上传播。本质上是说“这个错误已经被搞定了且应该被忽略”。它会阻止其它任何会被这个错误唤起的 errorCaptured 钩子和全局的 config.errorHandler

下面来看个例子

定义一个父组件cat

Vue.component('cat', {  
 template:`  
 <div>  
 <h1>Cat: </h1>  
 <slot></slot>  
 </div>`,  
 props:{  
 name:{  
 required:true,  
 type:String  
 }  
 },  
 errorCaptured(err,vm,info) {  
 console.log(`cat EC: ${err.toString()}\ninfo: ${info}`);   
 return false;  
 }  
  
});

定义一个子组件kitten，其中dontexist()并没有定义，存在错误

Vue.component('kitten', {  
 template:'<div><h1>Kitten: {{ dontexist() }}</h1></div>',  
 props:{  
 name:{  
 required:true,  
 type:String  
 }  
 }  
});

页面中使用组件

<div id="app" v-cloak>  
 <cat name="my cat">  
 <kitten></kitten>  
 </cat>  
</div>

在父组件的errorCaptured则能够捕获到信息

cat EC: TypeError: dontexist is not a function  
info: render

### 三、源码分析

异常处理源码

源码位置：/src/core/util/error.js

// Vue 全局配置,也就是上面的Vue.config  
import config from '../config'  
import { warn } from './debug'  
// 判断环境  
import { inBrowser, inWeex } from './env'  
// 判断是否是Promise，通过val.then === 'function' && val.catch === 'function', val ！=== null && val !== undefined  
import { isPromise } from 'shared/util'  
// 当错误函数处理错误时，停用deps跟踪以避免可能出现的infinite rendering  
// 解决以下出现的问题https://github.com/vuejs/vuex/issues/1505的问题  
import { pushTarget, popTarget } from '../observer/dep'  
  
export function handleError (err: Error, vm: any, info: string) {  
 // Deactivate deps tracking while processing error handler to avoid possible infinite rendering.  
 pushTarget()  
 try {  
 // vm指当前报错的组件实例  
 if (vm) {  
 let cur = vm  
 // 首先获取到报错的组件，之后递归查找当前组件的父组件，依次调用errorCaptured 方法。  
 // 在遍历调用完所有 errorCaptured 方法、或 errorCaptured 方法有报错时，调用 globalHandleError 方法  
 while ((cur = cur.$parent)) {  
 const hooks = cur.$options.errorCaptured  
 // 判断是否存在errorCaptured钩子函数  
 if (hooks) {  
 // 选项合并的策略，钩子函数会被保存在一个数组中  
 for (let i = 0; i < hooks.length; i++) {  
 // 如果errorCaptured 钩子执行自身抛出了错误，  
 // 则用try{}catch{}捕获错误，将这个新错误和原本被捕获的错误都会发送给全局的config.errorHandler  
 // 调用globalHandleError方法  
 try {  
 // 当前errorCaptured执行，根据返回是否是false值  
 // 是false，capture = true，阻止其它任何会被这个错误唤起的 errorCaptured 钩子和全局的 config.errorHandler  
 // 是true capture = fale，组件的继承或父级从属链路中存在的多个 errorCaptured 钩子，会被相同的错误逐个唤起  
 // 调用对应的钩子函数，处理错误  
 const capture = hooks[i].call(cur, err, vm, info) === false  
 if (capture) return  
 } catch (e) {  
 globalHandleError(e, cur, 'errorCaptured hook')  
 }  
 }  
 }  
 }  
 }  
 // 除非禁止错误向上传播，否则都会调用全局的错误处理函数  
 globalHandleError(err, vm, info)  
 } finally {  
 popTarget()  
 }  
}  
// 异步错误处理函数  
export function invokeWithErrorHandling (  
handler: Function,  
 context: any,  
 args: null | any[],  
 vm: any,  
 info: string  
 ) {  
 let res  
 try {  
 // 根据参数选择不同的handle执行方式  
 res = args ? handler.apply(context, args) : handler.call(context)  
 // handle返回结果存在  
 // res.\_isVue an flag to avoid this being observed，如果传入值的\_isVue为ture时(即传入的值是Vue实例本身)不会新建observer实例  
 // isPromise(res) 判断val.then === 'function' && val.catch === 'function', val ！=== null && val !== undefined  
 // !res.\_handled \_handle是Promise 实例的内部变量之一，默认是false，代表onFulfilled,onRejected是否被处理  
 if (res && !res.\_isVue && isPromise(res) && !res.\_handled) {  
 res.catch(e => handleError(e, vm, info + ` (Promise/async)`))  
 // avoid catch triggering multiple times when nested calls  
 // 避免嵌套调用时catch多次的触发  
 res.\_handled = true  
 }  
 } catch (e) {  
 // 处理执行错误  
 handleError(e, vm, info)  
 }  
 return res  
 }  
  
//全局错误处理  
function globalHandleError (err, vm, info) {  
 // 获取全局配置，判断是否设置处理函数，默认undefined  
 // 已配置  
 if (config.errorHandler) {  
 // try{}catch{} 住全局错误处理函数  
 try {  
 // 执行设置的全局错误处理函数，handle error 想干啥就干啥💗  
 return config.errorHandler.call(null, err, vm, info)  
 } catch (e) {  
 // 如果开发者在errorHandler函数中手动抛出同样错误信息throw err  
 // 判断err信息是否相等，避免log两次  
 // 如果抛出新的错误信息throw err Error('你好毒')，将会一起log输出  
 if (e !== err) {  
 logError(e, null, 'config.errorHandler')  
 }  
 }  
 }  
 // 未配置常规log输出  
 logError(err, vm, info)  
}  
  
// 错误输出函数  
function logError (err, vm, info) {  
 if (process.env.NODE\_ENV !== 'production') {  
 warn(`Error in ${info}: "${err.toString()}"`, vm)  
 }  
 /\* istanbul ignore else \*/  
 if ((inBrowser || inWeex) && typeof console !== 'undefined') {  
 console.error(err)  
 } else {  
 throw err  
 }  
}

### 小结

* handleError在需要捕获异常的地方调用，首先获取到报错的组件，之后递归查找当前组件的父组件，依次调用errorCaptured 方法，在遍历调用完所有 errorCaptured 方法或 errorCaptured 方法有报错时，调用 globalHandleError 方法
* globalHandleError调用全局的 errorHandler 方法，再通过logError判断环境输出错误信息
* invokeWithErrorHandling更好的处理异步错误信息
* logError判断环境，选择不同的抛错方式。非生产环境下，调用warn方法处理错误

## 参考文献

* https://juejin.cn/post/6844904096936230925
* https://segmentfault.com/a/1190000018606181