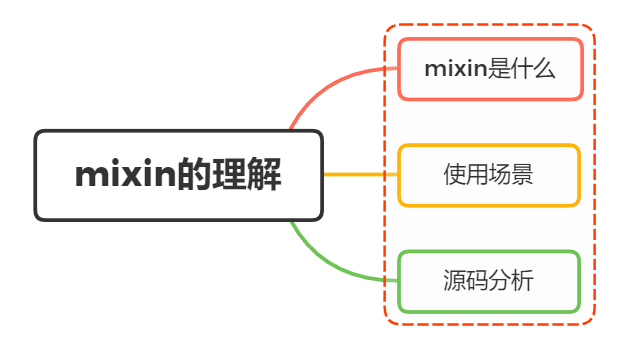
# 面试官：说说你对vue的mixin的理解，有什么应用场景？



## 一、mixin是什么

Mixin是面向对象程序设计语言中的类，提供了方法的实现。其他类可以访问mixin类的方法而不必成为其子类

Mixin类通常作为功能模块使用，在需要该功能时“混入”，有利于代码复用又避免了多继承的复杂

### Vue中的mixin

先来看一下官方定义 > mixin（混入），提供了一种非常灵活的方式，来分发 Vue 组件中的可复用功能。

本质其实就是一个js对象，它可以包含我们组件中任意功能选项，如data、components、methods、created、computed等等

我们只要将共用的功能以对象的方式传入 mixins选项中，当组件使用 mixins对象时所有mixins对象的选项都将被混入该组件本身的选项中来

在Vue中我们可以**局部混入**跟**全局混入**

### 局部混入

定义一个mixin对象，有组件options的data、methods属性

var myMixin = {  
 created: function () {  
 this.hello()  
 },  
 methods: {  
 hello: function () {  
 console.log('hello from mixin!')  
 }  
 }  
}

组件通过mixins属性调用mixin对象

Vue.component('componentA',{  
 mixins: [myMixin]  
})

该组件在使用的时候，混合了mixin里面的方法，在自动执行created生命钩子，执行hello方法

### 全局混入

通过Vue.mixin()进行全局的混入

Vue.mixin({  
 created: function () {  
 console.log("全局混入")  
 }  
})

使用全局混入需要特别注意，因为它会影响到每一个组件实例（包括第三方组件）

PS：全局混入常用于插件的编写

### 注意事项：

当组件存在与mixin对象相同的选项的时候，进行递归合并的时候组件的选项会覆盖mixin的选项

但是如果相同选项为生命周期钩子的时候，会合并成一个数组，先执行mixin的钩子，再执行组件的钩子

## 二、使用场景

在日常的开发中，我们经常会遇到在不同的组件中经常会需要用到一些相同或者相似的代码，这些代码的功能相对独立

这时，可以通过Vue的mixin功能将相同或者相似的代码提出来

举个例子

定义一个modal弹窗组件，内部通过isShowing来控制显示

const Modal = {  
 template: '#modal',  
 data() {  
 return {  
 isShowing: false  
 }  
 },  
 methods: {  
 toggleShow() {  
 this.isShowing = !this.isShowing;  
 }  
 }  
}

定义一个tooltip提示框，内部通过isShowing来控制显示

const Tooltip = {  
 template: '#tooltip',  
 data() {  
 return {  
 isShowing: false  
 }  
 },  
 methods: {  
 toggleShow() {  
 this.isShowing = !this.isShowing;  
 }  
 }  
}

通过观察上面两个组件，发现两者的逻辑是相同，代码控制显示也是相同的，这时候mixin就派上用场了

首先抽出共同代码，编写一个mixin

const toggle = {  
 data() {  
 return {  
 isShowing: false  
 }  
 },  
 methods: {  
 toggleShow() {  
 this.isShowing = !this.isShowing;  
 }  
 }  
}

两个组件在使用上，只需要引入mixin

const Modal = {  
 template: '#modal',  
 mixins: [toggle]  
};  
   
const Tooltip = {  
 template: '#tooltip',  
 mixins: [toggle]  
}

通过上面小小的例子，让我们知道了Mixin对于封装一些可复用的功能如此有趣、方便、实用

## 三、源码分析

首先从Vue.mixin入手

源码位置：/src/core/global-api/mixin.js

export function initMixin (Vue: GlobalAPI) {  
 Vue.mixin = function (mixin: Object) {  
 this.options = mergeOptions(this.options, mixin)  
 return this  
 }  
}

主要是调用merOptions方法

源码位置：/src/core/util/options.js

export function mergeOptions (  
 parent: Object,  
 child: Object,  
 vm?: Component  
): Object {  
  
if (child.mixins) { // 判断有没有mixin 也就是mixin里面挂mixin的情况 有的话递归进行合并  
 for (let i = 0, l = child.mixins.length; i < l; i++) {  
 parent = mergeOptions(parent, child.mixins[i], vm)  
 }  
}  
  
 const options = {}   
 let key  
 for (key in parent) {  
 mergeField(key) // 先遍历parent的key 调对应的strats[XXX]方法进行合并  
 }  
 for (key in child) {  
 if (!hasOwn(parent, key)) { // 如果parent已经处理过某个key 就不处理了  
 mergeField(key) // 处理child中的key 也就parent中没有处理过的key  
 }  
 }  
 function mergeField (key) {  
 const strat = strats[key] || defaultStrat  
 options[key] = strat(parent[key], child[key], vm, key) // 根据不同类型的options调用strats中不同的方法进行合并  
 }  
 return options  
}

从上面的源码，我们得到以下几点：

* 优先递归处理 mixins
* 先遍历合并parent 中的key，调用mergeField方法进行合并，然后保存在变量options
* 再遍历 child，合并补上 parent 中没有的key，调用mergeField方法进行合并，保存在变量options
* 通过 mergeField 函数进行了合并

下面是关于Vue的几种类型的合并策略

* 替换型
* 合并型
* 队列型
* 叠加型

### 替换型

替换型合并有props、methods、inject、computed

strats.props =  
strats.methods =  
strats.inject =  
strats.computed = function (  
 parentVal: ?Object,  
 childVal: ?Object,  
 vm?: Component,  
 key: string  
): ?Object {  
 if (!parentVal) return childVal // 如果parentVal没有值，直接返回childVal  
 const ret = Object.create(null) // 创建一个第三方对象 ret  
 extend(ret, parentVal) // extend方法实际是把parentVal的属性复制到ret中  
 if (childVal) extend(ret, childVal) // 把childVal的属性复制到ret中  
 return ret  
}  
strats.provide = mergeDataOrFn

同名的props、methods、inject、computed会被后来者代替

### 合并型

和并型合并有：data

strats.data = function(parentVal, childVal, vm) {   
 return mergeDataOrFn(  
 parentVal, childVal, vm  
 )  
};  
  
function mergeDataOrFn(parentVal, childVal, vm) {   
 return function mergedInstanceDataFn() {   
 var childData = childVal.call(vm, vm) // 执行data挂的函数得到对象  
 var parentData = parentVal.call(vm, vm)   
 if (childData) {   
 return mergeData(childData, parentData) // 将2个对象进行合并   
 } else {   
 return parentData // 如果没有childData 直接返回parentData  
 }  
 }  
}  
  
function mergeData(to, from) {   
 if (!from) return to   
 var key, toVal, fromVal;   
 var keys = Object.keys(from);   
 for (var i = 0; i < keys.length; i++) {  
 key = keys[i];  
 toVal = to[key];  
 fromVal = from[key];   
 // 如果不存在这个属性，就重新设置  
 if (!to.hasOwnProperty(key)) {  
 set(to, key, fromVal);  
 }   
 // 存在相同属性，合并对象  
 else if (typeof toVal =="object" && typeof fromVal =="object") {  
 mergeData(toVal, fromVal);  
 }  
 }   
 return to  
}

mergeData函数遍历了要合并的 data 的所有属性，然后根据不同情况进行合并：

* 当目标 data 对象不包含当前属性时，调用 set 方法进行合并（set方法其实就是一些合并重新赋值的方法）
* 当目标 data 对象包含当前属性并且当前值为纯对象时，递归合并当前对象值，这样做是为了防止对象存在新增属性

### 队列性

队列性合并有：全部生命周期和watch

function mergeHook (  
 parentVal: ?Array<Function>,  
 childVal: ?Function | ?Array<Function>  
): ?Array<Function> {  
 return childVal  
 ? parentVal  
 ? parentVal.concat(childVal)  
 : Array.isArray(childVal)  
 ? childVal  
 : [childVal]  
 : parentVal  
}  
  
LIFECYCLE\_HOOKS.forEach(hook => {  
 strats[hook] = mergeHook  
})  
  
// watch  
strats.watch = function (  
 parentVal,  
 childVal,  
 vm,  
 key  
) {  
 // work around Firefox's Object.prototype.watch...  
 if (parentVal === nativeWatch) { parentVal = undefined; }  
 if (childVal === nativeWatch) { childVal = undefined; }  
 /\* istanbul ignore if \*/  
 if (!childVal) { return Object.create(parentVal || null) }  
 {  
 assertObjectType(key, childVal, vm);  
 }  
 if (!parentVal) { return childVal }  
 var ret = {};  
 extend(ret, parentVal);  
 for (var key$1 in childVal) {  
 var parent = ret[key$1];  
 var child = childVal[key$1];  
 if (parent && !Array.isArray(parent)) {  
 parent = [parent];  
 }  
 ret[key$1] = parent  
 ? parent.concat(child)  
 : Array.isArray(child) ? child : [child];  
 }  
 return ret  
};

生命周期钩子和watch被合并为一个数组，然后正序遍历一次执行

### 叠加型

叠加型合并有：component、directives、filters

strats.components=  
strats.directives=  
  
strats.filters = function mergeAssets(  
 parentVal, childVal, vm, key  
) {   
 var res = Object.create(parentVal || null);   
 if (childVal) {   
 for (var key in childVal) {  
 res[key] = childVal[key];  
 }   
 }   
 return res  
}

叠加型主要是通过原型链进行层层的叠加

### 小结：

* 替换型策略有props、methods、inject、computed，就是将新的同名参数替代旧的参数
* 合并型策略是data, 通过set方法进行合并和重新赋值
* 队列型策略有生命周期函数和watch，原理是将函数存入一个数组，然后正序遍历依次执行
* 叠加型有component、directives、filters，通过原型链进行层层的叠加

## 参考文献

* https://zhuanlan.zhihu.com/p/31018570
* https://juejin.cn/post/6844904015495446536#heading-1
* https://juejin.cn/post/6844903846775357453
* https://vue3js.cn/docs/zh