题目

说下vue的响应式原理是如何实现的?

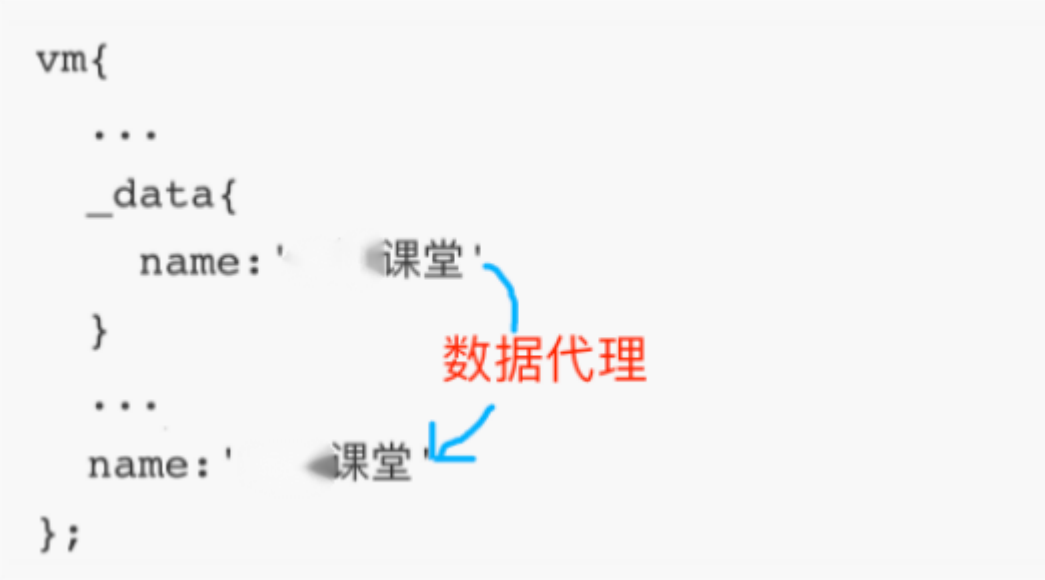
回答

- 思路
  - 原理（ Object.defineProperty() ）
    - 通过一个对象代理另一个对象属性的读写

```
Object.defineProperty('代理对象', '代理属性', {
  get() {              // getter   当读取'目标对象'的'代理属性'时，get函数/getter就会被调用，且返回代理属性的值
    return xxx;
  },
  set(value) {                 // setter   当修改'目标对象'的'代理属性'时，set函数/setter就会被调用，且收到修改的值
    xxx;
  },
});
```

```
let obj1 = {
  a: 111,
};
let obj2 = {};
Object.defineProperty(obj2, 'a', {
  get() {
    console.log('obj2被读取了');
    return obj1.a;
  },
  set(value) {
    console.log('obj2被修改了');
    obj1.a = value;
  },
});
```

  - Vue中应用的数据代理



- 通过vm对象属性代理 _data 中属性的读写
- 能更加方便的读写vue中data的数据
- 通过`Object.defineProperty()`把 data 中的属性添加到vm对象上，每个属性都有setter/getter

- 连环问
  - 说下vue3的响应式原理是如何实现的，为什么?
    - Object.defineProperty
      - 无法监听新增属性和删除属性，使用this.$set
      - 深层对象的劫持需要递归
      - 劫持数组时需要重写数组原生操作方法
      - 只是对对象的属性进行劫持
    - Proxy
      - 概述

        正如Proxy的英译"代理"所示，Proxy是ES6为了操作对象引入的API。它不直接作用在对象上，而是作为一种媒介，如果需要操作对象的话，需要经过这个媒介的同意。
      - 使用方式

```
let p = new Proxy(target, handler)
//target: 目标对象
//handler: 对对象进行拦截操作的函数，如set、get
```

      - 使用场景

```
const house = {
  name: '张三',
  price: '1000',
  phone: '18823139921',
  id: '111',
  state: '**',
};

const houseProxy = new Proxy(house, {
```

```
  // 读取代理
  get: function (target, key) {
    switch (key) {
      case 'phone':
        return '抱歉，不能告知'
      default:
        return Reflect.get(target,key);
    }
  },
  // 设置代理
  set: function (target, key, value) {
    if (key === 'id') {
      return Reflect.get(target,key);
    } else if (key === 'state') {
      return Reflect.set(target,key,value);
    }
  },
});

console.log(houseProxy.price);
console.log(houseProxy.phone);

houseProxy.id = '222';
houseProxy.state = '****';
console.log(houseProxy.id);
console.log(houseProxy.state);
```

- Reflect（映射）
  - 规范化、语义化
  - ES6规范中为了操作对象更加趋向于编程式（函数式）
  - 减少异常的抛出，因为对象的属性有可能是个getter或者setter
  - Reflect对象的方法与Proxy对象的方法一一对应