事件通信原理

Vue 中使用了大量的基于事件的发布/订阅通信，例如 event-bus：

- $on
- $emit

Vuex 容器内部也是基于此实现的。

```
const bus = new Vue()

// 监听一个自定义事件
bus.$on('事件类型', 处理函数)

// 发布事件
bus.$emit('事件类型', 处理函数)
```

```
const bus = new Vue()

bus.$on('a', () => {

})

bus.$on('b', () => {

})

bus.$emit('a')
```

```
function EventEmitter () {
  this.handlers = {}
}

// var obj = {}
// var a = 'abc'

// obj.a 访问名字叫 a 的成员
// obj['a'] 属性a
// obj[a] 变量 a
// obj['a'] [] 中的是 JavaScript 表达式，访问的是 [] 里面运算出来的结果
// obj['a' + 'b']

EventEmitter.prototype.on = function (eventName, callback) {
  let arr = this.handlers[eventName]

  if (!arr) {
    this.handlers[eventName] = []
    this.handlers[eventName].push(callback)
```

```
  } else {
    arr.push(callback)
  }
}

// 参数1：事件类型
// 参数2：所有的剩余参数
// 函数参数中使用的 ...args 称作 REST 参数
// args 是随便起的一个名字
// 它会把从第2个参数开始的所有后续参数收集到一个数组中
// 如果没有第2个参数，那它也是一个空数组
EventEmitter.prototype.emit = function (eventName, ...args) {
  const arr = this.handlers[eventName]
  if (arr) {
    arr.forEach(callback => {
      // 放到函数调用中的 ...数组，它的作用是展开数组
      // 所谓的展开就是把数组元素一个一个拿出来，传递给调用的函数
      // args[0], args[1], args[2]
      callback(...args)
    })
  }
}
```