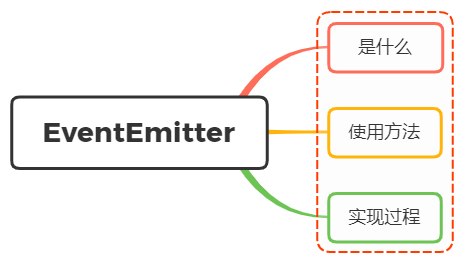
# 面试官：说说Node中的EventEmitter? 如何实现一个EventEmitter?



## 一、是什么

我们了解到，Node采用了事件驱动机制，而EventEmitter就是Node实现事件驱动的基础

在EventEmitter的基础上，Node几乎所有的模块都继承了这个类，这些模块拥有了自己的事件，可以绑定／触发监听器，实现了异步操作

Node.js 里面的许多对象都会分发事件，比如 fs.readStream 对象会在文件被打开的时候触发一个事件

这些产生事件的对象都是 events.EventEmitter 的实例，这些对象有一个 eventEmitter.on() 函数，用于将一个或多个函数绑定到命名事件上

## 二、使用方法

Node的events模块只提供了一个EventEmitter类，这个类实现了Node异步事件驱动架构的基本模式——观察者模式

在这种模式中，被观察者(主体)维护着一组其他对象派来(注册)的观察者，有新的对象对主体感兴趣就注册观察者，不感兴趣就取消订阅，主体有更新的话就依次通知观察者们

基本代码如下所示：

const EventEmitter = require('events')  
  
class MyEmitter extends EventEmitter {}  
const myEmitter = new MyEmitter()  
  
function callback() {  
 console.log('触发了event事件！')  
}  
myEmitter.on('event', callback)  
myEmitter.emit('event')  
myEmitter.removeListener('event', callback);

通过实例对象的on方法注册一个名为event的事件，通过emit方法触发该事件，而removeListener用于取消事件的监听

关于其常见的方法如下：

* emitter.addListener/on(eventName, listener) ：添加类型为 eventName 的监听事件到事件数组尾部
* emitter.prependListener(eventName, listener)：添加类型为 eventName 的监听事件到事件数组头部
* emitter.emit(eventName[, ...args])：触发类型为 eventName 的监听事件
* emitter.removeListener/off(eventName, listener)：移除类型为 eventName 的监听事件
* emitter.once(eventName, listener)：添加类型为 eventName 的监听事件，以后只能执行一次并删除
* emitter.removeAllListeners([eventName])： 移除全部类型为 eventName 的监听事件

## 三、实现过程

通过上面的方法了解，EventEmitter是一个构造函数，内部存在一个包含所有事件的对象

class EventEmitter {  
 constructor() {  
 this.events = {};  
 }  
}

其中events存放的监听事件的函数的结构如下：

{  
 "event1": [f1,f2,f3]，  
 "event2": [f4,f5]，  
 ...  
}

然后开始一步步实现实例方法，首先是emit，第一个参数为事件的类型，第二个参数开始为触发事件函数的参数，实现如下：

emit(type, ...args) {  
 this.events[type].forEach((item) => {  
 Reflect.apply(item, this, args);  
 });  
}

当实现了emit方法之后，然后实现on、addListener、prependListener这三个实例方法，都是添加事件监听触发函数，实现也是大同小异

on(type, handler) {  
 if (!this.events[type]) {  
 this.events[type] = [];  
 }  
 this.events[type].push(handler);  
}  
  
addListener(type,handler){  
 this.on(type,handler)  
}  
  
prependListener(type, handler) {  
 if (!this.events[type]) {  
 this.events[type] = [];  
 }  
 this.events[type].unshift(handler);  
}

紧接着就是实现事件监听的方法removeListener/on

removeListener(type, handler) {  
 if (!this.events[type]) {  
 return;  
 }  
 this.events[type] = this.events[type].filter(item => item !== handler);  
}  
  
off(type,handler){  
 this.removeListener(type,handler)  
}

最后再来实现once方法， 再传入事件监听处理函数的时候进行封装，利用闭包的特性维护当前状态，通过fired属性值判断事件函数是否执行过

once(type, handler) {  
 this.on(type, this.\_onceWrap(type, handler, this));  
 }  
  
 \_onceWrap(type, handler, target) {  
 const state = { fired: false, handler, type , target};  
 const wrapFn = this.\_onceWrapper.bind(state);  
 state.wrapFn = wrapFn;  
 return wrapFn;  
 }  
  
 \_onceWrapper(...args) {  
 if (!this.fired) {  
 this.fired = true;  
 Reflect.apply(this.handler, this.target, args);  
 this.target.off(this.type, this.wrapFn);  
 }  
 }

完整代码如下：

class EventEmitter {  
 constructor() {  
 this.events = {};  
 }  
  
 on(type, handler) {  
 if (!this.events[type]) {  
 this.events[type] = [];  
 }  
 this.events[type].push(handler);  
 }  
  
 addListener(type,handler){  
 this.on(type,handler)  
 }  
  
 prependListener(type, handler) {  
 if (!this.events[type]) {  
 this.events[type] = [];  
 }  
 this.events[type].unshift(handler);  
 }  
  
 removeListener(type, handler) {  
 if (!this.events[type]) {  
 return;  
 }  
 this.events[type] = this.events[type].filter(item => item !== handler);  
 }  
  
 off(type,handler){  
 this.removeListener(type,handler)  
 }  
  
 emit(type, ...args) {  
 this.events[type].forEach((item) => {  
 Reflect.apply(item, this, args);  
 });  
 }  
  
 once(type, handler) {  
 this.on(type, this.\_onceWrap(type, handler, this));  
 }  
  
 \_onceWrap(type, handler, target) {  
 const state = { fired: false, handler, type , target};  
 const wrapFn = this.\_onceWrapper.bind(state);  
 state.wrapFn = wrapFn;  
 return wrapFn;  
 }  
  
 \_onceWrapper(...args) {  
 if (!this.fired) {  
 this.fired = true;  
 Reflect.apply(this.handler, this.target, args);  
 this.target.off(this.type, this.wrapFn);  
 }  
 }  
}

测试代码如下：

const ee = new EventEmitter();  
  
// 注册所有事件  
ee.once('wakeUp', (name) => { console.log(`${name} 1`); });  
ee.on('eat', (name) => { console.log(`${name} 2`) });  
ee.on('eat', (name) => { console.log(`${name} 3`) });  
const meetingFn = (name) => { console.log(`${name} 4`) };  
ee.on('work', meetingFn);  
ee.on('work', (name) => { console.log(`${name} 5`) });  
  
ee.emit('wakeUp', 'xx');  
ee.emit('wakeUp', 'xx'); // 第二次没有触发  
ee.emit('eat', 'xx');  
ee.emit('work', 'xx');  
ee.off('work', meetingFn); // 移除事件  
ee.emit('work', 'xx'); // 再次工作

## 参考文献

* http://nodejs.cn/api/events.html#events\_class\_eventemitter
* https://segmentfault.com/a/1190000015762318
* https://juejin.cn/post/6844903781230968845
* https://vue3js.cn/interview