【前端面试】使用JS实现一个EventBus自定...

前言

EventBus是事件总线的意思，可不是什么**事件车。**事件总线模式在工作中经常使用，在面试中也很容易问到。甚至在很多面试中会让你手写一个EventBus，那么EventBus到底是个什么东西，今天我们就来学一学！

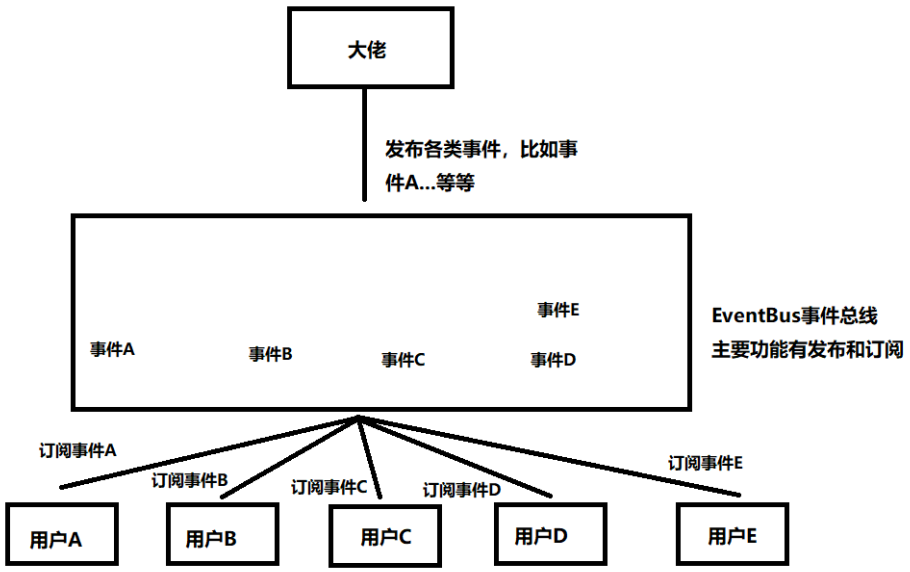
1.什么是EventBus?

所谓事件总线模式，其实就和发布订阅模式非常类似，比如我们订阅了一个公众号，公众号发布文章之后我们就能收到信息，这就是一种订阅发布的关系。

再比如在Vue项目中，我们可以使用$on、$emit来实现事件的监听和触发，这其实就是一种事件总线的思想在里面，只不过Vue帮我们实现好了。

在我们的JavaScript中，可以给元素添加一个点击监听事件，当用户点击的时候，点击事件怎会被执行，这也是一种事件总线的思想在里面，就好比元素订阅了点击事件，用户发布或出触发点击事件。

从上可以看出，事件总线模式在我们的开发中经常出现，我们也可以通过一张图来更加清楚的认识什么是事件总线。



上图很清晰的描述了EventBus的角色是什么，它主要承担的是一个发布订阅的责任，比如有人发布了信息出来，那其他人如何接收这些信息呢？其他人可以通过EventBus来订阅这些信息，当有信息发布的时候，事件总线就将这些信息传播给订阅者。

2.乞丐版EventBus

既然我们了解了EventBus的原理后，我们便可以手动实现一个来。我们步子不能迈的太大，先来一个乞丐版，实现嘴贱的发布订阅功能。

实现目标：

- 使用$on订阅事件
- 使用$emit发布事件

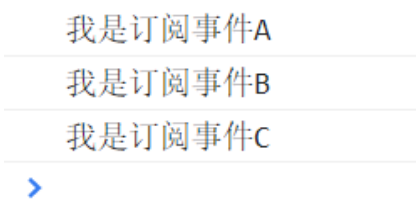
代码示例：

```
<script>
  class EventBus {
    // 定义所有事件列表,格式如下：
    // {
    //   key: Array,
    //   key: Array,
    // }
    // Array存储的是注册的回调函数
    constructor() {
      this.eventObj = {}; // 用于存储所有订阅事件
    }
    // 订阅事件,类似监听事件$on('key',()=>{})
    $on(name, callbcak) {
      // 判断是否存储过
      if(!this.eventObj[name]) {
        this.eventObj[name] = [];
      }
      this.eventObj[name].push(callbcak); // 往事件数组里面push
    }
    // 发布事件,类似于触发事件$emit('key')
    $emit(name) {
      // 获取存储的事件回调函数数组
      const eventList = this.eventObj[name];
      // 执行所有回调函数
      for (const callbcak of eventList) {
        callbcak();
      }
    }
  }
  // 初始化EventBus
  let EB = new EventBus();
  // 订阅事件
  EB.$on('key1', () => {
    console.info("我是订阅事件A");
```

```
  })
  EB.$on("key1", () => {
    console.info("我是订阅事件B");
  })
  EB.$on("key2", () => {
    console.info("我是订阅事件C");
  })

  // 发布事件
  EB.$emit('key1');
  EB.$emit('key2');
</script>
```

输出结果：



上段代码中我们声明了一个EventBus类，专门用来处理我们的发布订阅操作。上段代码的整体

思路如下：

1. 首先订阅了一堆事件key1、key2等等，当这些事件被触发时执行回调函数。
2. 订阅的这些事件key1...都需要存储到EventBus中去，定义变量eventObj存储。
3. 当其它用户或模块触发订阅的事件key1...等，EventBus就去eventObj中查找，找到则则触发存储的回调函数。

上面就是一个最简单的EventBus了，只初步实现了$on和$emit。

3.传参版EventBus

虽然说乞丐版的EventBus也能用，但是它的场景很有限。我们使用EventBus是很多时候都是需要传参的，就好比我订阅了一个公众号，结果公众号每次发布的内容都是空的，很明显不行。所以我们需要在发布事件$emit的时候传参，然后在订阅事件的回调函数里面可以接收参数。

实现目标：

- 使用$on订阅事件
- 使用$emit发布事件
- $emit发布事件可以传参

示例代码：

```
<script>
  class EventBus {
    // 定义所有事件列表,格式如下：
    // {
    //   key: Array,
```

```
    //   key: Array,
    // }
    // Array存储的是注册的回调函数
    constructor() {
      this.eventObj = {}; // 用于存储所有订阅事件
    }
    // 订阅事件,类似监听事件$on('key',()=>{})
    $on(name, callbcak) {
      // 判断是否存储过
      if (!this.eventObj[name]) {
        this.eventObj[name] = [];
      }
      this.eventObj[name].push(callbcak); // 往事件数组里面push
    }
    // 发布事件,类似于触发事件$emit('key')
    $emit(name, ...args) {
      // 获取存储的事件回调函数数组
      const eventList = this.eventObj[name];
      // 执行所有回调函数且传入参数
      for (const callbcak of eventList) {
        callbcak(...args);
      }
    }
  }
  // 初始化EventBus
  let EB = new EventBus();

  // 订阅事件
  EB.$on('key1', (name, age) => {
    console.info("我是订阅事件A:", name, age);
  })
  EB.$on("key1", (name, age) => {
    console.info("我是订阅事件B:", name, age);
  })
  EB.$on("key2", (name) => {
    console.info("我是订阅事件C:", name);
  })

  // 发布事件
  EB.$emit('key1', "小猪课堂", 26);
  EB.$emit('key2', "小猪课堂");
</script>
```

输出结果：

```
我是订阅事件A: 小猪课堂 26
我是订阅事件B: 小猪课堂 26
我是订阅事件C: 小猪课堂
>
```

上段代码我们实现了$emit传参，$on的回调可以接收参数的功能。这样我们的EventBus又变得完善了一点。比如有两个标签页需要通信，就可以使用我们这里的EventBus来进行实现。除此之外，Vue组件间的通讯也可以使用EventBus来进行实现，因为可以传递参数。

4.取消订阅版EventBus

既然有订阅，那么就有取消订阅，我们可以订阅公众号，也可以取关公众号，这是理所当然的事。我们的EventBus也需要实现这样的功能，比如我们监听了某个事件，在一定情况下我们需要取消监听事件。

实现目标：

- 使用$on订阅事件
- 使用$emit发布事件
- $emit发布事件可以传参
- 实现$off取消订阅

示例代码：

```
<script>
  class EventBus {
    // 定义所有事件列表,此时需要修改格式：
    // // {
    //   key: {
    //     id: Function,
    //     id: Function
    //   },
    //   key: Object,
    // }
    // Array存储的是注册的回调函数
    constructor() {
      this.eventObj = {}; // 用于存储所有订阅事件
      this.callbcakId = 0; // 每个函数的ID
    }
    // 订阅事件,类似监听事件$on('key',()=>{})
    $on(name, callbcak) {
      // 判断是否存储过
      if (!this.eventObj[name]) {
        this.eventObj[name] = {};
      }
      // 定义当前回调函数id
      const id = this.callbcakId++;
      this.eventObj[name][id] = callbcak; // 以键值对的形式存储回调函数
      return id; // 将id返回出去，可以利用该id取消订阅
    }
    // 发布事件,类似于触发事件$emit('key')
    $emit(name, ...args) {
      // 获取存储的事件回调函数数组
```

```
      const eventList = this.eventObj[name];
      // 执行所有回调函数且传入参数
      for (const id in eventList) {
        eventList[id](...args);
      }
    }
    // 取消订阅函数，类似于$off('key1', id)
    $off(name, id) {
      // 删除存储在事件列表中的该事件
      delete this.eventObj[name][id];
      console.info(`id为${id}的事件已被取消订阅`)
      // 如果这是最后一个订阅者，则删除整个对象
      if (!Object.keys(this.eventObj).length) {
        delete this.eventObj[name];
      }
    }
  }
  // 初始化EventBus
  let EB = new EventBus();

  // 订阅事件
  EB.$on('key1', (name, age) => {
    console.info("我是订阅事件A:", name, age);
  })
  let id = EB.$on("key1", (name, age) => {
    console.info("我是订阅事件B:", name, age);
  })
  EB.$on("key2", (name) => {
    console.info("我是订阅事件C:", name);
  })

  // 发布事件key1
  EB.$emit('key1', "小猪课堂", 26);
  // 取消订阅事件
  EB.$off('key1', id);
  // 发布事件key1
  EB.$emit('key1', "小猪课堂", 26);
  // 发布事件
  EB.$emit('key2', "小猪课堂");
</script>
```

输出结果：

```
我是订阅事件A: 小猪课堂 26
我是订阅事件B: 小猪课堂 26
id为1的事件已被取消订阅
我是订阅事件A: 小猪课堂 26
我是订阅事件C: 小猪课堂
>
```

既然我们需要取消订阅某一个事件，那么我们就需要给该事件添加一个标识，这样才可以在茫茫人海中找到它，并且取阅它。所以我们这个修改了eventObj事件对象的存储结构，其中的每个key存储的不在是数组了，而是对象类型。

实现的整体思路如下：

1. 给每个订阅事件添加唯一标识id。
2. 之前事件key存储的是事件回调函数数组，现在改成事件回调函数对象，键为id，值为回调函数。
3. 订阅事件的时候返回一个id。
4. 取消订阅的时候通过id找到存储在eventObj中的事件函数，并且删掉它。

5.执行一次版EventBus(完整版)

虽然前面实现的EventBus基本能够满足我们的项目需求了，但是还有一种情景我们需要考虑，比如我订阅了某个公众号，但是我只允许你给我发送一次消息，然后我们取关你。虽然在公众号的场景下这种需求比较变态，但是在我们的项目场景下这可能就是比较正常的了，我只想让某一个订阅事件只执行一次，这非常正常。

- 使用$on订阅事件
- 使用$emit发布事件
- $emit发布事件可以传参
- 实现$off取消订阅
- 实现$once执行一次

示例代码：

```
<script>
  class EventBus {
    // 定义所有事件列表,此时需要修改格式：
    // // {
    //   key: {
    //     D+id: Function,
    //     id: Function
    //   },
    //   key: Object,
    // }
    // Array存储的是注册的回调函数
    constructor() {
      this.eventObj = {}; // 用于存储所有订阅事件
      this.callbcakId = 0; // 每个函数的ID
    }
    // 订阅事件,类似监听事件$on('key',()=>{})
    $on(name, callbcak) {
      // 判断是否存储过
```

```
      if (!this.eventObj[name]) {
        this.eventObj[name] = {};
      }
      // 定义当前回调函数id
      const id = this.callbcakId++;
      this.eventObj[name][id] = callbcak; // 以键值对的形式存储回调函数
      return id; // 将id返回出去，可以利用该id取消订阅
    }
    // 发布事件,类似于触发事件$emit('key')
    $emit(name, ...args) {
      // 获取存储的事件回调函数数组
      const eventList = this.eventObj[name];
      // 执行所有回调函数且传入参数
      for (const id in eventList) {
        eventList[id](...args);
        // 如果是订阅一次，则删除
        if(id.indexOf('D') !== -1) {
          delete eventList[id];
        }
      }
    }
    // 取消订阅函数，类似于$off('key1', id)
    $off(name, id) {
      console.log(this.eventObj)
      // 删除存储在事件列表中的该事件
      delete this.eventObj[name][id];
      console.info(`${id}id事件已被取消订阅`)
      // 如果这是最后一个订阅者，则删除整个对象
      if (!Object.keys(this.eventObj).length) {
        delete this.eventObj[name];
      }
    }
    // 订阅事件，只会执行一次，为了方便，id上直接加上一个标识d
    $once(name, callbcak){
      // 判断是否存储过
      if (!this.eventObj[name]) {
        this.eventObj[name] = {};
      }
      // 定义当前回调函数id,添加D则代表只执行一次
      const id = "D" + this.callbcakId++;
      this.eventObj[name][id] = callbcak; // 以键值对的形式存储回调函数
      return id; // 将id返回出去，可以利用该id取消订阅
    }
  }
  // 初始化EventBus
  let EB = new EventBus();

  // 订阅事件
  EB.$on('key1', (name, age) => {
    console.info("我是订阅事件A:", name, age);
  })
```

```
  EB.$once("key1", (name, age) => {
    console.info("我是订阅事件B:", name, age);
  })
  EB.$on("key2", (name) => {
    console.info("我是订阅事件C:", name);
  })

  // 发布事件key1
  EB.$emit('key1', "小猪课堂", 26);
  console.info("在触发一次key1")
  EB.$emit('key1', "小猪课堂", 26);
  // 发布事件
  EB.$emit('key2', "小猪课堂");
</script>
```

输出结果：

```
我是订阅事件A: 小猪课堂 26
我是订阅事件B: 小猪课堂 26
在触发一次key1
我是订阅事件A: 小猪课堂 26
我是订阅事件C: 小猪课堂
>
```

上段代码中我们第一次触发事件key1时，订阅事件A和B都执行了，但是第二次触发事件key1时，只有订阅事件A执行了，说明订阅事件B已经取消订阅了。

整体实现思路：

1. 使用$once订阅事件时，我们需要给该事件做个标识，以便识别出它只会执行一次。
2. 为了方便，我们直接在id字段前面拼接D代表只会执行一次。
3. 触发事件时，判断id上是否有D，如果有D，则执行完后删除该事件。

总结

本篇文章实现了一个较为简单的EventBus，但是也基本满足我们在项目中的使用了。想要实现EventBus，我们首要理解的就是发布订阅模式，然后我们在思考下面几个问题，基本上就能实现一个属于自己的EventBus了。

- 订阅事件如何存储?
- 如何传递参数?
- 如何给每个订阅事件添加唯一标识?
- 如何正确删除存储的订阅事件?
- $on和$emit主要担任的是什么角色?
- 什么时发布订阅模式?

想明白了上面的问题那么EventBus我相信你也明白了，能够自己实现了。