30 天精通RxJS (26)：简易实作Observable(一)

Jan 11th, 2017 . 4 mins read

> 因为实在太多读者在问要如何实作Observable，所以特别调整了本系列文章最后几篇的内容，空出一天的位置来写如何简易实作Observable。

为什么是简易实作而不完整实作呢？当然这个系列的文章是希望读者能学会如何使用RxJS，而实作Observable 其实只是帮助我们理解Observable 的运作方式，所以这篇文章会尽可能地简单，一来让读者容易理解及吸收，二来有兴趣的读者可以再沿着这篇文章的内容去完整的实作。

重点观念

Observable跟Observer Pattern是不同的，Observable内部并没有管理一份订阅清单，**订阅Observable就像是执行一个function一样**！

所以实作过程的重点

- 订阅就是执行一个funciton
- 订阅接收的物件具备next, error, complete 三个方法
- 订阅会返回一个可退订(unsubscribe)的物件

基本observable 实作

先用最简单的function 来建立observable 物件

```
function create(subscriber) {
    var observable = {
        subscribe: function(observer) {
            subscriber(observer)
        }
    };
```

```
function create(subscriber) {
    var observable = {
        subscribe: function(observer) {
            subscriber(observer)
        }
    };
    return observable;
}

var observable = create(function(observer) {
  observer.next(1);
  observer.next(2);
  observer.next(3);
})

var observer = {
  next: function(value) {
    console.log(value)
  }
}

observable.subscribe(observer)
// 1
// 2
// 3
```

JSBin

这时我们已经有最简单的功能了，但这里有一个大问题，就是observable 在结束(complete)就不应该再发送元素

```
var observable = create(function(observer) {
  observer.next(1);
  observer.next(2);
  observer.next(3);
  observer.complete();
  observer.next('still work');
})
```





```
  },
  complete: function() {
    console.log('complete!')
  }
}

observable.subscribe(observer)
// 1
// 2
// 3
// "complete!"
// "still work"
```

JSBin

从上面的程式码可以看到complete 之后还是能送元素出来，另外还有一个问题就是observer，如果是不完整的就会出错，这也不是我们希望看到的。

```
var observable = create(function(observer) {
  observer.next(1);
  observer.next(2);
  observer.next(3);
  observer.complete(); // error: complete is not a function
})

var observer = {
  next: function(value) {
    console.log(value)
  }
}

observable.subscribe(observer)
// 1
// 2
// 3
// "complete!"
// "still work"
```

实作简易Observer

要修正这两个问题其实并不难，我们只要实作一个Observer 的类别，每次使用者传入的observer都会利用这个类别转乘我们想要Observer 物件。

首先订阅时有可能传入一个observer 物件，或是一到三个function(next, error, complete)，所以我们要建立一个类别可以接受各种可能的参数

```
class Observer {
  constructor(destinationOrNext, error, complete) {
    switch (arguments.length) {
      case 0:
        // 空的 observer
      case 1:
        if (!destinationOrNext) {
          // 空的 observer
        }
        if (typeof destinationOrNext === 'object') {
          // 傳入了 observer 物件
        }
      default:
        // 如果上面都不是，代表應該是傳入了一到三個 function
        break;
    }
  }
}
```

写一个方法(safeObserver)来回传正常的observer

```
class Observer {
  constructor(destinationOrNext, error, complete) {
    // ... 一些程式碼
  }
  safeObserver(observerOrNext, error, complete) {
    let next;
```

```
    } else if (observerOrNext) {
      // observerOrNext 是 observer 物件
      next = observerOrNext.next || () => {};
      error = observerOrNext.error || function(err) {
        throw err
      };
      complete = observerOrNext.complete || () => {};
    }
    // 最後回傳我們預期的 observer 物件
    return {
      next: next,
      error: error,
      complete: complete
    };
  }
}
```

再把constructor 完成

```
// 預設空的 observer
const emptyObserver = {
  next: () => {},
  error: (err) => { throw err; },
  complete: () => {}
}

class Observer {
  constructor(destinationOrNext, error, complete) {
    switch (arguments.length) {
      case 0:
        // 空的 observer
        this.destination = this.safeObserver(emptyObserver);
        break;
      case 1:
        if (!destinationOrNext) {
          // 空的 observer
          this.destination = this.safeObserver(emptyObserver);
          break;
```

```
            this.destination = this.safeObserver(destinationOrNext);
            break;
          }
        default:
          // 如果上面都不是，代表應該是傳入了一到三個 function
          this.destination = this.safeObserver(destinationOrNext, error,
          break;
      }
    }
    safeObserver(observerOrNext, error, complete) {
      // ... 一些程式碼
    }
  }
```

JSBin

这里我们把真正的observer塞到 `this.destination` ，接着完成observer的方法。

Observer 的三个主要的方法(next, error, complete)都应该结束或退订后不能再被执行，所以我们在物件内部偷塞一个boolean 值来作为是否曾经结束的依据。

```
class Observer {
  constructor(destinationOrNext, error, complete) {
    // ... 一些程式碼
  }
  safeObserver(observerOrNext, error, complete) {
    // ... 一些程式碼
  }
  unsubscribe() {
    this.isStopped = true; // 偷塞一個屬性 isStopped
  }
}
```

接着要实作三个主要的方法就很简单了，只要先判断 `isStopped` 在使用 `this.destination` 物件来传送值就可以了

```
class Observer {
  constructor(destinationOrNext, error, complete) {
```

```
  }

  next(value) {
    if (!this.isStopped && this.next) {
      // 先判斷是否停止過
      try {
        this.destination.next(value); // 傳送值
      } catch (err) {
        this.unsubscribe();
        throw err;
      }
    }
  }

  error(err) {
    if (!this.isStopped && this.error) {
      // 先判斷是否停止過
      try {
        this.destination.error(err); // 傳送錯誤
      } catch (anotherError) {
        this.unsubscribe();
        throw anotherError;
      }
      this.unsubscribe();
    }
  }

  complete() {
    if (!this.isStopped && this.complete) {
      // 先判斷是否停止過
      try {
        this.destination.complete(); // 發送停止訊息
      } catch (err) {
        this.unsubscribe();
        throw err;
      }
      this.unsubscribe(); // 發送停止訊息後退訂
    }
```





```
    this.isStopped = true;
  }
}
```

JSBin

到这里我们就完成基本的Observer 实作了，接着让我们拿到基本版的observable 中使用吧。

```
function create(subscriber) {
    const observable = {
        subscribe: function(observerOrNext, error, complete) {
            const realObserver = new Observer(observerOrNext, error, c
            subscriber(realObserver);
            return realObserver;
        }
    };
    return observable;
}

var observable = create(function(observer) {
  observer.next(1);
  observer.next(2);
  observer.next(3);
  observer.complete();
  observer.next('not work');
})

var observer = {
  next: function(value) {
    console.log(value)
  },
  complete: function() {
      console.log('complete!')
  }
}

observable.subscribe(observer);
// 1
```

JSBin

到这里我们就完成最基本的observable 了，至少基本的行为都跟我们期望的一致，我知道读者们仍然不会放过我，你们会希望做出一个Observable 型别以及至少一个operator 对吧？不用担心，我们下一篇就会讲解如何建立一个Observable 型别和operator 的方法！

今日小结

今天我们复习了Observable 的重要概念，并用这些重要的概念实作出了基本的observable 以及Observer 的类别。

不知道今天读者们有没有收获呢？如果有任何问题，欢迎在下方留言给我，谢谢！

Tags

JavaScript RxJS Observable Observer RxJS 30 Days

Prev Next