# 8.3 事件总线

在APP中，我们经常会需要一个广播机制，用以跨页面事件通知，比如一个需要登录的APP中，页面会关注用户登录或注销事件，来进行一些状态更新。这时候，一个事件总线便会非常有用，事件总线通常实现了订阅者模式，订阅者模式包含发布者和订阅者两种角色，可以通过事件总线来触发事件和监听事件，本节我们实现一个简单的全局事件总线，我们使用单例模式，代码如下：

//订阅者回调签名  
typedef void EventCallback(arg);  
  
class EventBus {  
 //私有构造函数  
 EventBus.\_internal();  
  
 //保存单例  
 static EventBus \_singleton = new EventBus.\_internal();  
  
 //工厂构造函数  
 factory EventBus()=> \_singleton;  
  
 //保存事件订阅者队列，key:事件名(id)，value: 对应事件的订阅者队列  
 var \_emap = new Map<Object, List<EventCallback>>();  
  
 //添加订阅者  
 void on(eventName, EventCallback f) {  
 if (eventName == null || f == null) return;  
 \_emap[eventName] ??= new List<EventCallback>();  
 \_emap[eventName].add(f);  
 }  
  
 //移除订阅者  
 void off(eventName, [EventCallback f]) {  
 var list = \_emap[eventName];  
 if (eventName == null || list == null) return;  
 if (f == null) {  
 \_emap[eventName] = null;  
 } else {  
 list.remove(f);  
 }  
 }  
  
 //触发事件，事件触发后该事件所有订阅者会被调用  
 void emit(eventName, [arg]) {  
 var list = \_emap[eventName];  
 if (list == null) return;  
 int len = list.length - 1;  
 //反向遍历，防止在订阅者在回调中移除自身带来的下标错位   
 for (var i = len; i > -1; --i) {  
 list[i](arg);  
 }  
 }  
}  
  
//定义一个top-level变量，页面引入该文件后可以直接使用bus  
var bus = new EventBus();

使用

//页面A中  
...  
 //监听登录事件  
bus.on("login", (arg) {  
 // do something  
});  
  
//登录页B中  
...  
//登录成功后触发登录事件，页面A中订阅者会被调用  
bus.emit("login", userInfo);

注意：Dart中实现单例模式的标准做法就是使用static变量+工厂构造函数的方式，这样就可以保证new EventBus()始终返回都是同一个实例，读者应该理解并掌握这种方法。

事件总线通常用于组件之间状态共享，但关于组件之间状态共享也有一些专门的包如redux、以及前面介绍过的Provider。对于一些简单的应用，事件总线是足以满足业务需求的，如果你决定使用状态管理包的话，一定要想清楚您的APP是否真的有必要使用它，防止“化简为繁”、过度设计。