```
//1、事件类型
public enum EVENT_TYPE
{
	//通用
	RELOAD_UI,
	RELOAD_POPUP,
	FORBID_INTERACTION,
	ALLOW_INTERACTION,

	//场景加载
	SCENE_LOADED,
};

//2、事件系统管理器
public class EventManager : MonoBehaviour
{

	#region C# properties		// 单例模式
	public static EventManager Instance {
		get { return instance;}
		set { }
	}
	#endregion

	#region variables
	private static EventManager instance = null;

	// 定义一个字典存储事件的监听队列
	private Dictionary<EVENT_TYPE, List<IListener>> listenersDic = new Dictionary<EVENT_TYPE,
List<IListener>> ();
	#endregion

	// 初始化单例
	void Awake() {
		if (instance == null) {
			instance = this;
			DontDestroyOnLoad (gameObject);
		} else {
			DestroyImmediate (this);
		}
	}
	#region Methods
	/// <summary>
	/// 添加事件监听者到监听队列
```

```
    /// </summary>
    /// <param name="eventType">Event to Listen for.  监听的事件类型</param>
    /// <param name="listener">Object to listen for event.</param>
    public void AddListener(EVENT_TYPE eventType, IListener listener) {
        //  事件监听队列
        List<IListener> listenList = null;

        //  判断是否存在该事件类型 key，若存在将该监听者添加至监听队列
        if (listenersDic.TryGetValue(eventType, out listenList)) {
            listenList.Add(listener);
            return;
        }

        //  若不存在该事件类型 key，创建一个新的监听队列后添加该监听者，最后存储至
字典中
        listenList = new List<IListener>();
        listenList.Add (listener);
        listenersDic.Add (eventType, listenList);
    }

    /// <summary>
    ///  向监听者发送事件消息
    /// </summary>
    /// <param name="eventType">Event to invoke.</param>
    /// <param name="sender">Object invoking event.</param>
    /// <param name="param">Optional argument.</param>
    public void PostNotification(EVENT_TYPE eventType, Component sender, UnityEngine.Object
param = null) {
        //  获取事件的监听队列，若不存在直接返回
        List<IListener> listenList = null;
        if (!listenersDic.TryGetValue (eventType, out listenList)) {
            return;
        }

        //  若存在，进行通知
        for (int i = 0; i < listenList.Count; i++) {
            if (!listenList [i].Equals (null)) {
                listenList [i].OnEvent (eventType, sender, param);
            }
        }
    }

    //  移除某一事件的监听
    public void RemoveEvent(EVENT_TYPE eventType) {
        listenersDic.Remove(eventType);
    }

    // 移除 null 监听（场景切换时部分监听者被销毁）
    public void RemoveRendundancies() {
        Dictionary<EVENT_TYPE,    List<IListener>>    tmpListenersDic    =    new
Dictionary<EVENT_TYPE, List<IListener>>();

        foreach(KeyValuePair<EVENT_TYPE, List<IListener>> item in listenersDic) {
            // 检测监听队列中所有项，移除 null 项
            for (int i=item.Value.Count - 1; i>=0;i--) {
                if (item.Value[i].Equals(null)) {
                    item.Value.RemoveAt (i);
                }
            }

            //移除所有 null 后，若不为空添加至临时字典
            if (item.Value.Count > 0)
                tmpListenersDic.Add (item.Key, item.Value);
        }

        listenersDic = tmpListenersDic;;
    }

    // 切换场景时移除 null 监听
    void OnEnable()
    {
        SceneManager.sceneLoaded += OnSceneLoaded;
    }
    void OnDisable()
    {
        SceneManager.sceneLoaded -= OnSceneLoaded;
    }
    void OnSceneLoaded(Scene scence, LoadSceneMode mod)
    {
        RemoveRendundancies();
    }
    #endregion
}

//3、监听者接口
public interface IListener
{
    void  OnEvent(EVENT_TYPE  eventType,  Component  sender,  UnityEngine.Object  param  =
null);
}

//4、传递参数（可自定义）
public class PassInt : UnityEngine.Object
{
        public int value;

        public PassInt(int _i)
        {
                value = _i;
        }
}
```