**概念**

总线：负责发送命令、事件、持久化Saga

命令：描述一个命令

事件：描述一个事件

处理器：处理命令或事件

Saga：代表一个流程，是由完成一个流程的一系列处理器组成

**什么是Saga**

Saga代表一个流程

如下订单流程，下订单包含如下命令和事件：

下订单命令：

用于开启saga流程，此时saga被创建并持久化（如保存到数据库中），用户点击“下单”时，发送一个Http请求触发该命令，该命令会发送“检查商品库存命令”

检查商品库存命令：

这里会检查商品库存，如果库存充足，则创建“订单聚合”，状态为未支付，并结束Http请求

支付完成事件：

如果用户支付完成，会触发该事件，更改“订单聚合”，状态为以支付，并发送“订单创建完成事件”

此时saga以完成，将会被移除

同理配送流程：

订单创建完成事件：

开启saga流程，该事件会创建配送单

......

**Saga与处理器**

Saga是执行上有顺序的步骤集合，处理器是一个步骤

应用场景示例：

Saga：下订单Saga，用于下订单

处理器：订单创建完成事件处理器，用于记录谁在什么时候下了什么订单

**代码示例**

命令和事件：

// 表示一个命令或事件

public abstract class Message {

    public Guid SagaId { get; }

    public Message (Guid sagaId) {

        SagaId = sagaId;

    }

}

// 下订单命令

public class PlaceOrderCommand : Message {

    public PlaceOrderCommand () : base (Guid.NewGuid ()) { }

}

...

Saga：

public abstract class SagaBase {

    public SagaBase () {

        SagaId = Guid.NewGuid ();

    }

public Guid SagaId { get; set; }

    public bool IsCompleted { get; set; }

}

public class PlaceOrderSaga : SagaBase,

    IHandler<PlaceOrderCommand>,

    IHandler<CheckInStoreCommand>,

    IHandler<PayCompletedEnvet> {

    static PlaceOrderSaga () {

        Bus.RegisterSaga<PlaceOrderCommand, PlaceOrderSaga> ();

}

    public void Handle (PlaceOrderCommand message) {

        // 发送检查库存命令

        Bus.Send (new CheckInStoreCommand (SagaId));

    }

    public void Handle (CheckInStoreCommand message) {

        /\* 检查库存 \*/

        /\* 如果库存充足，传教订单聚合 \*/

    }

    public void Handle (PayCompletedEnvet message) {

        // 发送订单完成事件

        Bus.Send (new OrderCompletedEvent (SagaId));

        // saga 完成

        IsCompleted = true;

    }

}

总线：

public class Bus {

    // 注册的处理器

    private static Dictionary<Type, Type> \_handlers = new Dictionary<Type, Type> ();

    // 注册的 saga 类型

private static Dictionary<Type, Type> \_sagaTypes = new Dictionary<Type, Type> ();

    // 已启动的 saga（saga应该保存到数据库中，但这里为了方便）

private static Dictionary<Guid, SagaBase> \_sagaInstances = new Dictionary<Guid, SagaBase> ();

    // 注册 Saga

    public static void RegisterSaga<TMessage, TSaga> () where TMessage : Message where TSaga : SagaBase {

        \_sagaTypes.Add (typeof (TMessage), typeof (TSaga));

}

    // 注册 处理器

    public static void RegisterHandler<TMessage, THandler> () where TMessage : Message where THandler : IHandler<TMessage> {

        \_handlers.Add (typeof (TMessage), typeof (THandler));

}

    // 发送事件或命令

    public static void Send<T> (T message) where T : Message {

        // 运行以注册的 处理器

        if (\_handlers.ContainsKey (typeof (T))) {

            var handlerType = \_handlers[typeof (T)];

            var handler = (IHandler<T>) Activator.CreateInstance (handlerType);

            handler.Handle (message);

        }

        // message 是否可以启动一个 saga

        if (\_sagaTypes.ContainsKey (typeof (T))) {

            var sagaType = \_sagaTypes[typeof (T)];

            var saga = (SagaBase) Activator.CreateInstance (sagaType);

            var handler = (IHandler<T>) saga;

            handler.Handle (message);

            // 将 saga 持久化

            \_sagaInstances.Add (saga.SagaId, saga);

            return;

        }

        // 查看已启动的 Saga 中是否存在对应的 Saga

        if (\_sagaInstances.ContainsKey (message.SagaId)) {

            var saga = \_sagaInstances[message.SagaId];

            var handler = (IHandler<T>) saga;

            handler.Handle (message);

            // 如果该 saga 以完成，则移除

            if (saga.IsCompleted) {

                \_sagaInstances.Remove (message.SagaId);

                return;

            }

            // 否则更新 saga 的状态到数据库

            return;

        }

    }

}